

Remarks


Despite the Applicant's 37 CFR §1.131 declaration filed in the last response, claims 1-24 remain rejected in view of Mathon et al. The Office Action alleges that the declaration is ineffective to overcome the reference because there was not a sufficient nexus established in the submitted evidence and discussion to the claims in the instant application.

In this reply, Applicant submits supplemental declaration by the inventor, supporting documentation that includes excerpts from the computer source code implementing the invention and mapping of the code to each of the claims. The supporting documentation also includes test data results from a run of that computer code. Applicant believes that the evidence and discussion submitted herewith establish sufficient nexus to the claims.

The claims are believed to be patentable and a favorable Office Action is hereby earnestly solicited. If a telephone interview would be of assistance in advancing prosecution of the subject application, the Examiner is requested to telephone the number provided below. Please charge any fee due in connection with this reply to deposit account no. 02-0393 of Baker & McKenzie LLP.

Date: January 13, 2006

Respectfully submitted,


Eunhee Park
Registration No. 42,976
BAKER & McKENZIE
805 Third Avenue
New York, NY 10022
(212) 751-5700 telephone
(212) 759-9133 facsimile

Mapping of Claims to Source Code and Data Results

The following document associates each of the claims with the modules and/or functional line numbers for the "Message Tracking Monitor" (mtrkmon) software.

The software module that implements the invention is called the "message tracking monitor". The software source tree for the module is as follows:

```
./builder/apps/mtrkmon/mtrkmon.c
./builder/apps/mtrkmon/mtrkmon_app.c
./builder/apps/mtrkmon/mtrkmon_app.h
./builder/apps/mtrkmon/mtrk_dsn.c
./builder/apps/mtrkmon/mtrk_dsn.h
./builder/apps/mtrkmon/mtrk_mdn.c
./builder/apps/mtrkmon/mtrk_mdn.h
./builder/apps/mtrkmon/Product.ebi
./builder/doc/builder_notes.txt
./builder/doc/builder_roadmap.txt
./builder/doc/builder_summary.txt
./builder/doc/man/mail_api.txt
./builder/doc/man/mail_cb.txt
./builder/doc/man/mm_api.txt
./builder/doc/man/sec_api.txt
./builder/doc/manuals/sdk_ref/appendix_a.fm
./builder/doc/manuals/sdk_ref/appendix_a_del.fm
./builder/doc/manuals/sdk_ref/appendix_b.fm
./builder/doc/manuals/sdk_ref/components.fm
./builder/doc/manuals/sdk_ref/c_ref/c++_ref.book
./builder/doc/manuals/sdk_ref/c_ref/c++_ref.fm
./builder/doc/manuals/sdk_ref/c_ref/c++_reftoc.fm
./builder/doc/manuals/sdk_ref/c_ref/front_matter.fm
./builder/doc/manuals/sdk_ref/c_ref/preface.fm
./builder/doc/manuals/sdk_ref/front_matter.fm
./builder/doc/manuals/sdk_ref/glossary.fm
./builder/doc/manuals/sdk_ref/intro.fm
./builder/doc/manuals/sdk_ref/mdbuilder.book
./builder/doc/manuals/sdk_ref/mdbuilderix.fm
./builder/doc/manuals/sdk_ref/mdbuildertoc.fm
./builder/doc/manuals/sdk_ref/preface.fm
./builder/doc/manuals/sdk_ref/process.fm
./builder/doc/manuals/sdk_ref/tcl_ref/front_matter.fm
./builder/doc/manuals/sdk_ref/tcl_ref/preface.fm
./builder/doc/manuals/sdk_ref/tcl_ref/tcl_ref.book
./builder/doc/manuals/sdk_ref/tcl_ref/tcl_ref.fm
./builder/doc/manuals/sdk_ref/tcl_ref/tcl_reftoc.fm
```

./builder/doc/manuals/sdk_ref/ui.fm
./builder/mail/cache.c
./builder/mail/cache.h
./builder/mail/decode.c
./builder/mail/decode.h
./builder/mail/imap.c
./builder/mail/imap.h
./builder/mail/imap_--.msg
./builder/mail/lsearch.c
./builder/mail/lsearch.h
./builder/mail/mail.c
./builder/mail/mail.h
./builder/mail/mailfile.c
./builder/mail/mailfile.h
./builder/mail/mailgets.c
./builder/mail/mailgets.h
./builder/mail/mailhook.c
./builder/mail/mailparam.c
./builder/mail/mailutil.c
./builder/mail/mailutil.h
./builder/mail/mail_--.msg
./builder/mail/mh.c
./builder/mail/mh.h
./builder/mail/mh_cache.c
./builder/mail/mh_cache.h
./builder/mail/mh_ndb.c
./builder/mail/mh_ndb.h
./builder/mail/misc.c
./builder/mail/misc.h
./builder/mail/pcache.c
./builder/mail/pcache.h
./builder/mail/pop3.c
./builder/mail/pop3.h
./builder/mail/pop3_--.msg
./builder/mail/Product.ebi
./builder/mail/search.c
./builder/mail/search.h
./builder/mail/sm.c
./builder/mail/sm.h
./builder/mail/sm_util.c
./builder/mail/sm_util.h
./builder/mail/src.mac
./builder/mail/src.mac/ca_util.c
./builder/mail/src.mac/mailutil_plt.c
./builder/mail/src.mac/mh_plt.c
./builder/mail/src.mac/mh_plt.h

./builder/mail/src.unx
./builder/mail/src.unx/ca_util.c
./builder/mail/src.unx/mailutil_plt.c
./builder/mail/src.unx/mh_plt.c
./builder/mail/src.unx/mh_plt.h
./builder/mail/src.win
./builder/mail/src.win/ca_util.c
./builder/mail/src.win/mailutil_plt.c
./builder/mail/src.win/mh_plt.c
./builder/mail/src.win/mh_plt.h
./builder/maildata
./builder/maildata/charset.c
./builder/maildata/charset.h
./builder/maildata/fstring.c
./builder/maildata/fstring.h
./builder/maildata/maildata.c
./builder/maildata/maildata.h
./builder/maildata/mailstring.h
./builder/maildata/mime.c
./builder/maildata/mime.h
./builder/maildata/Product.ebi
./builder/maildata/rfc822.c
./builder/maildata/rfc822.h
./builder/mm
./builder/mm/hdrutil.c
./builder/mm/hdrutil.h
./builder/mm/mm.c
./builder/mm/mm.h
./builder/mm/mmcalls.c
./builder/mm/mmcalls.h
./builder/mm/mmfol.c
./builder/mm/mmint.h
./builder/mm/mmms.c
./builder/mm/mmmsg.c
./builder/mm/mmnot.c
./builder/mm/mmsec.c
./builder/mm/mmsec.h
./builder/mm/mmutil.c
./builder/mm/mmutil.h
./builder/mm/mm_--.msg
./builder/mm/Product.ebi
./builder/monitor/glue/mdb_plug.c
./builder/monitor/glue/mdb_plug.h
./builder/monitor/glue/README.txt
./builder/monitor/mmonitor.c
./builder/monitor/mmonitor.h

./builder/monitor/mmonitor_int.h
./builder/monitor/mmonitor_process.c
./builder/monitor/mmonitor_th.c
./builder/monitor/mmon_--.msg
./builder/monitor/Product.ebi
./builder/mtrk/api/mtrk.h
./builder/mtrk/api/Product.ebi
./builder/mtrk/db/mtrkdb.c
./builder/mtrk/db/mtrkdb.h
./builder/mtrk/db/mtrkdb32.dsp
./builder/mtrk/db/mtrkdb32.dsw
./builder/mtrk/db/mtrkdbth.dsp
./builder/mtrk/db/mtrkdb_access.c
./builder/mtrk/db/mtrkdb_access.h
./builder/mtrk/db/mtrkdb_table.c
./builder/mtrk/db/mtrkdb_table.h
./builder/mtrk/db/mtrkdb_test.c
./builder/mtrk/db/mtrkdb_utility.c
./builder/mtrk/db/mtrkdb_utility.h
./builder/mtrk/db/Product.ebi
./builder/mtrk/Product.ebi
./builder/mtrk/report/mtrkrep.c
./builder/mtrk/report/mtrkrep.h
./builder/mtrk/report/mtrkrep32.dsp
./builder/mtrk/report/mtrkrep32.dsw
./builder/mtrk/report/mtrkrepth.dsp
./builder/mtrk/report/mtrkrep_test.c
./builder/mtrk/report/Product.ebi
./builder/Product.ebi
./builder/sdk_c/mbuilder.c
./builder/sdk_c/mbuilder.def
./builder/sdk_c/mbuilder.h
./builder/sdk_c/mbuilder32.dsp
./builder/sdk_c/mb_--.msg
./builder/sdk_c/mb_env.h
./builder/sdk_c/mb_glue.c
./builder/sdk_c/mb_plug.c
./builder/sdk_c/mb_plug.h
./builder/sdk_c/mb_ver.h
./builder/sdk_c/mb_verenv.c
./builder/sdk_c/mem_c_api.c
./builder/sdk_c/mem_c_api.h
./builder/sdk_c/mrep_file.c
./builder/sdk_c/mrep_file.h
./builder/sdk_c/Product.ebi
./builder/sdk_c/src.unx/mb_env.c

```
./builder/sdk_c/src.unx/mb_pathenv.c
./builder/sdk_c/src.win/mb_env.c
./builder/sdk_c/src.win/mb_pathenv.c
./builder/sdk_c/test
./builder/sdk_c/test/datafiles/camping.jpg
./builder/sdk_c/test/datafiles/CVS
./builder/sdk_c/test/datafiles/CVS/Entries
./builder/sdk_c/test/datafiles/CVS/Repository
./builder/sdk_c/test/datafiles/CVS/Root
./builder/sdk_c/test/datafiles/CVS/Tag
./builder/sdk_c/test/datafiles/message.txt
./builder/sdk_c/test/datafiles/message2.txt
./builder/sdk_c/test/datafiles/message3.txt
./builder/sdk_c/test/datafiles/message4.txt
./builder/sdk_c/test/mb_c_test.c
./builder/sdk_c/test/Product.ebi
```

This represents a complete, standalone, fully functional service for tracking messages delivered to the Internet email system. The software that implements the tracking components was first derived in prototype form for a desktop email client consisting of the message tracking MIME parsing utilities described in:

Following are specific code filename, line number and header history notice references, including automatic timestamp references generated by our CVS (Code Version System) source code management system. Source code tracking for the entire builder system extends to back to when the module was first created from an even earlier version of electronic mail API (c-client). These are direct excerpts from the source files.

Claims

1. A message tracking system, comprising: a message tracking monitor operable to monitor one or more ports for tracking information; a message tracking interface coupled to the message tracking monitor and enabled to communicate with one or more decision support subsystems, the message tracking interface operable to receive tracking information from the message tracking monitor and transfer the tracking information to one or more decision support subsystems, wherein the tracking information is collected and managed.

The message tracking monitor command is encapsulated in the file:

```
builder/apps/mtrkmon/mtrkmon.c
```

/* COPYRIGHT

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

```
* All rights reserved.  
*  
* Acquisition and use of this software and related materials for any  
* purpose requires a written license agreement from MessagingDirect Limited,  
* or a written license from an organization licensed by MessagingDirect Limited  
* to grant such a license.  
* END COPYRIGHT */
```

```
/* OVERVIEW
```

```
* This module contains the Message Tracking Monitor (mtrkmon) daemon command  
function  
* definitions. The mtrkmon daemon is an MBuilder Monitor application that monitors  
* a set of mailboxes looking for message tracking status messages. Specifically it  
* looks for Delivery Status Notifications (DSN) and Message Disposition Notifications  
* (MDN). Upon receipt of these messages it, it parses their content and updates  
* the Message Tracking Database (mtrkdb) with new status information.  
* END OVERVIEW */
```

The generic mail monitoring subsystem is encapsulated in the file:

`builder/monitor/mmonitor.c:`

```
/* COPYRIGHT  
* Copyright (c) MessagingDirect Limited, Edmonton, Canada  
* All rights reserved.  
*  
* Acquisition and use of this software and related materials for any  
* purpose requires a written license agreement from MessagingDirect Limited,  
* or a written license from an organization licensed by MessagingDirect Limited  
* to grant such a license.  
* END COPYRIGHT */  
  
/* OVERVIEW  
* This module contains definitions for the functions (methods) for the message  
* monitor API.  
* END OVERVIEW */
```

The message tracking database (decision support database) is encapsulated in the file:

`builder/mtrk/db/mtrkdb.c`

```
/* COPYRIGHT  
* Copyright (c) MessagingDirect Limited, Edmonton, Canada  
* All rights reserved.  
*
```

* Acquisition and use of this software and related materials for any
* purpose requires a written license agreement from MessagingDirect Limited,
* or a written license from an organization licensed by MessagingDirect Limited
* to grant such a license.
* END COPYRIGHT */

/* OVERVIEW

* This module contains the main function points for the Message Tracking database.
* END OVERVIEW */

The message tracking event (notification posting) is encapsulated in the file:

builder/mtrk/report/mtrkrep.c

/* COPYRIGHT

* Copyright (c) MessagingDirect Limited, Edmonton, Canada
* All rights reserved.
*
* Acquisition and use of this software and related materials for any
* purpose requires a written license agreement from MessagingDirect Limited,
* or a written license from an organization licensed by MessagingDirect Limited
* to grant such a license.
* END COPYRIGHT */

/* OVERVIEW

* This module contains the main API points for the Message Tracking Reporting
subsystem.
* END OVERVIEW */

**2. The message tracking system as claimed in claim 1, wherein the one or more
ports include any one or combination of electronic mail address, message
tracking server, access status log, and proprietary message tracking API.**

The supported tracking "ports" as supported by the implementation include:

Standards based Internet email tracking events:

builder/apps/mtrkmon/mtrk_dsn.c - Delivery Status Notifications
builder/apps/mtrkmon/mtrk_mdnc.c - Message Disposition Notifications

/* COPYRIGHT

* Copyright (c) MessagingDirect Limited, Edmonton, Canada
* All rights reserved.
*
* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,
* or a written license from an organization licensed by MessagingDirect Limited
* to grant such a license.
* END COPYRIGHT */

/* OVERVIEW

* This module contains routines to parse and manage rfc1894 Delivery Status
* Notification content.
* END OVERVIEW */

/* COPYRIGHT

* Copyright (c) MessagingDirect Limited, Edmonton, Canada
* All rights reserved.
*
* Acquisition and use of this software and related materials for any
* purpose requires a written license agreement from MessagingDirect Limited,
* or a written license from an organization licensed by MessagingDirect Limited
* to grant such a license.
* END COPYRIGHT */

/* OVERVIEW

* This module contains routines to parse and manage rfc2298 Message Disposition
* Notification content.
* END OVERVIEW */

Programatic tracking events:

builder/apps/mtrk/api/mtrk.h - C/C++ tracking event posting

Log based tracking events:

builder/apps/mtrkmon/mtrkmon.c - File based log monitoring and parsing

3. The message tracking system as claimed in claim 1, wherein the decision support subsystem includes a decision support database.

Provided in file:

builder/mtrk/db/mtrkdb.c

4. The message tracking system as claimed in claim 1, wherein the decision support subsystem includes a decision support application.

Provided in file:

builder/apps/mtrkmon/mtrkmon.c - List and report all tracking events

5. The message tracking system as claimed in claim 1, wherein the tracking information includes one or more tracking notifications.

Standards based Internet email tracking notifications collected as DSN or MDN messages:

builder/apps/mtrkmon/mtrk_dsn.c - Delivery Status Notifications

builder/apps/mtrkmon/mtrk_mdn.c - Message Disposition Notifications

6. The message tracking system as claimed in claim 1, wherein the message tracking interface is operable to access the tracking information stored in a decision support database.

Provided in file:

builder/mtrk/db/mtrkdb.c

7. The message tracking system as claimed in claim 1, wherein the message tracking interface is operable to update the tracking information stored in a decision support database.

Provided in file:

builder/mtrk/db/mtrkdb.c

8. The message tracking system as claimed in claim 1, wherein the message tracking interface is operable to manage the message tracking monitor.

Provided in file:

builder/apps/mtrkmon/mtrkmon.c - Monitor mailbox for tracking notifications

9. The message tracking system as claimed in claim 1, wherein the system further includes one or more dynamically pluggable decision support interfaces.

Provided in file:

builder/mtrk/db/mtrkdb.h

/* COPYRIGHT

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

```

* All rights reserved.
*
* Acquisition and use of this software and related materials for any
* purpose requires a written license agreement from MessagingDirect Limited,
* or a written license from an organization licensed by MessagingDirect Limited
* to grant such a license.
* END COPYRIGHT */

```

```

/* OVERVIEW

```

```

* This module contains the public API for the Message Tracking database.
* END OVERVIEW */

```

The message tracking database provides a "pluggable" interface in the form of C driver structures. The structure consists of an interface definition and one or more implementations in the form a list or vector of C function pointers. Message tracking database API instantiates the driver for the database or other decision support system based on configuration and/or tracking event type.

10. A method for tracking message, comprising: monitoring tracking information on one or more ports; collecting tracking information from the one or more ports; and transmitting the tracking information for handling the tracking information according to a predetermined schema.

Provided in the files:

```

builder/apps/mtrkmon/mtrkmon.c - message tracking monitor application
builder/mtrk/db/mtrkdb.c       - message tracking database (including schema)

```

11. The method for tracking messages as claimed in claim 10, wherein the one or more ports include one or more electronic message tracking ports.

As in claim 2.

12. The method for tracking messages as claimed in claim 10, further including: parsing the tracking information into a record format.

Provided in file:

```

builder/mtrk/db/mtrkdb.h (excerpt lines 88 - 126)

```

```

/* message status codes */
typedef enum {
    MTRK_SC_INVALID = 0,          /* so that zero is not a valid status code */

```

```

        MTRK_SC_SUBMITTED,                /* message has been injected into
mail transfer system */
        MTRK_SC_RELAYED,                  /* message transferred from one host
to another */
        MTRK_SC_DELIVERED,                /* message has reached the
destination mailbox */
        MTRK_SC_FORWARDED,                /* message has been forwarded to a
mail client (ie: mail list exploder */
        MTRK_SC_BLACKHOLE,                /* message has been relayed to a
non-DSN capable host */
        MTRK_SC_DELAYED,                  /* message has been temporarily
delayed at the specified host */
        MTRK_SC_REJECTED,                 /* message not able to be delivered
to destination mailbox */
        MTRK_SC_SEEN                      /* message has been opened by user using
an MUA */
    } MTRK_STATUS_CODE_t;

#define MTRK_STATUS_CODE_FIRST MTRK_SC_SUBMITTED /* for validity
checking of status codes */
#define MTRK_STATUS_CODE_LAST MTRK_SC_SEEN

/* to convert message status codes to strings */
#define MTRK_SC_SUBMITTED_STRING "Submitted"
#define MTRK_SC_RELAYED_STRING "Relayed"
#define MTRK_SC_DELIVERED_STRING "Delivered"
#define MTRK_SC_FORWARDED_STRING "Forwarded"
#define MTRK_SC_BLACKHOLE_STRING "Non DSN Aware"
#define MTRK_SC_DELAYED_STRING "Delayed"
#define MTRK_SC_REJECTED_STRING "Rejected"
#define MTRK_SC_SEEN_STRING "Seen"
#define MTRK_SC_INVALID_STRING "Not recognized"

/* message status record layout */
typedef struct {
    char * msgid;                          /* unique message identifier */
    MTRK_STATUS_CODE_t status;              /* status of message */
    char * host;                           /* host to which status record applies */
    char * to_host;                         /* used only by RELAYED state records */
    char * user;                           /* used only by SEEN state records */
    char * description;                    /* status reason description (optional) */
    time_t timestamp;                      /* creation timestamp for status record */
} MTRK_STATUS_REC_t;

```

13. The method for tracking messages as claimed in claim 10, wherein the

predetermined schema includes storing in a decision support database.

Provided in file:

builder/mtrk/db/mtrkdb.h

14. The method for tracking messages as claimed in claim 10, wherein the predetermined schema includes real time exception handling.

The mtrkmon reporting interface includes support for registering external notification (callback handlers) for real time event processing and handling. These are bound to PHP or Tcl "handler" functions for processing real time exceptions based on event keys.

Provided in file:

builder/mtrk/db/mtrkrep.h (excerpt line 56 - 83)

```
/* PHP callback function types */

/* : record match callback */
typedef int (*MTRKREP_PHP_REC_MATCH_CB_ft) ( /* R: zero if successful, non-zero if not */
/* D: callback function to handle a
record match */
void * full_key, /* I: full_key of matching record */
char ** record_string, /* I: formatted record */
void * data /* I: callback context data */
);

/* : key match callback */
typedef int (*MTRKREP_PHP_KEY_MATCH_CB_ft) ( /* R: zero if successful, non-zero if not */
/* D: callback function to handle a
key match */
char * key_value, /* I: full_key which matched */
char * key_recno, /* I: alphanumeric representation of the recno */
int key_length, /* I: the length of the key returned */
void * data /* I: callback context data */
);
```

15. The method for tracking messages as claimed in claim 10, wherein the transmitting includes transmitting the tracking information to a decision support subsystem.

Provided in file:

`builder/mtrk/db/mtrkdb.c`

16. The method for tracking messages as claimed in claim 10, wherein the one or more ports include one or more tracking ports.

As in claim 2.

17. The method for tracking messages as claimed in claim 10, further including providing analytical reports based on the collected tracking information.

Provided in file:

`builder/mtrk/report/mtrkrep.c`

18. The method for tracking messages as claimed in claim 10, wherein the monitoring includes requesting for one or more tracking notifications from one or more tracking sources.

Provided in file:

`builder/apps/mtrkmon/mtrkmon.c` (excerpt lines 60-74)

The following shows the command environment structure which includes variable (configurable) monitoring of multiple locations.

```
/* : DOC - the command environment structure. All of the information used to
   : control the operation of the application taken from the command environment
   : is held here. Sources of information include the command line arguments,
   : and the OS environment. */
typedef struct command_environment {
    char * config_file;          /* pathname of config file */
    BOOLEAN debug_mode;          /* are we in debugging mode */
    BOOLEAN interactive_mode;    /* run in daemon mode or not */
    char * database_dir;         /* message tracking database directory path */
    int n_mailboxes;             /* number of mailboxes to monitor */
    char ** mailboxes;           /* list of mailboxes to monitor */
    char *db_path;               /* path to message tracking database */
} CENV_ENV_t;
```

19. The method for tracking messages as claimed in claim 10, wherein the monitoring includes accepting one or more tracking notifications from one or more tracking sources.

As in claim 18.

20. The method for tracking messages as claimed in claim 10, wherein the monitoring includes continuously monitoring tracking information ports.

As in claim 18. Specifically, when run as a "daemon" the monitor will continuously poll tracking information sources until told to stop.

21. A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps of tracking messages, comprising: monitoring tracking information on one or more ports; collecting tracking information from the one or more ports; and transmitting the tracking information for handling the tracking information according to a predetermined schema.

As in claim 1.

22. A message tracking system, comprising: a message tracking monitor coupled to one or more message tracking data sources, the message tracking monitor perable to receive tracking information from the one or more message tracking data sources and parse the tracking information into one or more tracking data records; a message tracking interface coupled to the message tracking monitor and one or more decision support subsystems, the message tracking interface operable to receive tracking data records and transmit the tracking data records to the one or more decision support subsystems.

As in claim 1.

23. The message tracking system as claimed in claim 22, further including: one or more dynamically pluggable decision support interface modules coupled to the message tracking interface and the one or more decision support subsystems, wherein the one or more dynamically pluggable decision support interface modules in response to receiving the one or more tracking data records from the message tracking interface, transmit the one or more tracking data records to the one or more decision support subsystems.

As in claim 1 and claim 9 combined.

24. The message tracking system as claimed in claim 22, wherein the tracking information includes message transactions on the Internet.

As in claim 1 and claim 9 combined.

The following pages show output or data results of the source code above:

Debug: *****

Debug: ***** Processing Folder: 'neg_match'

Note: Total number of messages processed: 6

Note: - DSN reports processed: 0

Note: - MDN reports processed: 0

Note: - Exchange reports processed: 0

Note: - Qmail reports processed: 0

Note: Number of unprocessable messages: 6

Note: Percentage of messages processed as reports: %0

Debug: *****

Debug: ***** Processing Folder: 'pos_match_neg_content'

Note: Total number of messages processed: 1

Note: - DSN reports processed: 0

Note: - MDN reports processed: 0

Note: - Exchange reports processed: 0

Note: - Qmail reports processed: 0

Note: Number of unprocessable messages: 1

Note: Percentage of messages processed as reports: %0

Debug: *****

Debug: ***** Processing Folder: 'pos_match_pos_content'

Note: Total number of messages processed: 12

Note: - DSN reports processed: 3

Note: - MDN reports processed: 2

Note: - Exchange reports processed: 5

Note: - Qmail reports processed: 2

Note: Number of unprocessable messages: 0

Note: Percentage of messages processed as reports: %100

Debug: *****

Debug: ***** Parsing Folder: 'pos_match_pos_content' *****

Debug: ***** Parsing Message: '0' *****

Debug: Detected standard DSN report type.

Action: relayed

Original-Envelope-Id:

<MBUILDER.1020423110011.24.27296@msgsyd1.emailbill.hpa.com.au>

Reporting-MTA: dns;msgsyd2.emailbill.hpa.com.au

Arrival-Date: Tue, 23 Apr 2002 21:56:47 +1000

Original-Recipient: rfc822;belven@bigpond.com.au

Final-Recipient: rfc822;belven@bigpond.com.au

Status: 2.0.0

Remote-MTA: DNS;extmail.bigpond.com

Diagnostic-Code: SMTP;250 ok

Last-Attempt-Date: Tue, 23 Apr 2002 22:09:58 +1000

Debug: ***** Parsing Message: '1' *****

Debug: Detected Microsoft Exchange delivery report type.

Action: delivered

Original-Envelope-Id: <MGEN.1030717151436.0.22652@ecdev.fnbo.com>

Reporting-MTA: Exchange;fnni.com

Original-Recipient: rfc822;dkirschner@fnni.com

Status: 2.0.0

Debug: ***** Parsing Message: '2' *****

Debug: Detected standard DSN report type.

Action: failed

Original-Envelope-Id:

<MBUILDER.1020423110009.2.27296@msgsyd1.emailbill.hpa.com.au>

Reporting-MTA: dns;msgsyd2.emailbill.hpa.com.au

Arrival-Date: Tue, 23 Apr 2002 21:56:48 +1000

Original-Recipient: rfc822;mmitre10@mccays

Final-Recipient: rfc822;mmitre10@mccays

Status: 5.1.1

Remote-MTA: DNS;mail.mccays.com.au

Diagnostic-Code: SMTP;550 Relaying is prohibited

Last-Attempt-Date: Tue, 23 Apr 2002 22:05:03 +1000

Debug: ***** Parsing Message: '3' *****

Debug: Detected Qmail delivery report type.

Action: failed

Original-Envelope-Id:

MBUILDER.1020423110011.27.27296@msgsyd1.emailbill.hpa.com.au

Reporting-MTA: Qmail;mail.iinet.net.au

Original-Recipient: rfc822;christer@iinet.net.au
Status: 5.0.0
Diagnostic-Code: smtp;Sorry, no mailbox here by that name. 5.1.1

Debug: ***** Parsing Message: '4' *****
Debug: Detected standard DSN report type.
Action: failed
Original-Envelope-Id:
<MBUILDER.1020423110010.12.27296@msgsyd1.emailbill.hpa.com.au>
Reporting-MTA: dns;msgsyd2.emailbill.hpa.com.au
Arrival-Date: Tue, 23 Apr 2002 21:56:46 +1000
Original-Recipient: rfc822;gerry@shahingrop.com.au
Final-Recipient: rfc822;gerry@shahingrop.com.au
Status: 5.1.2
Remote-MTA: DNS;shahingrop.com.au
Last-Attempt-Date: Tue, 23 Apr 2002 22:04:07 +1000

Debug: ***** Parsing Message: '5' *****
Debug: Detected Qmail delivery report type.
Action: failed
Original-Envelope-Id:
MBUILDER.1020502110010.44.6538@msgsyd1.emailbill.hpa.com.au
Reporting-MTA: Qmail;mail.tsn.cc
Original-Recipient: rfc822;jhatppep@tsn.cc
Status: 5.0.0
Diagnostic-Code: smtp;Sorry, no mailbox here by that name. vpopmail 5.1.1

Debug: ***** Parsing Message: '6' *****
Debug: Detected standard MDN report type.
Disposition: displayed
Original-Message-Id:
<MBUILDER.1020422110007.215.1880@msgsyd1.emailbill.hpa.com.au>
Final-Recipient: rfc822;d.wahlquist@latrobe.edu.au

Debug: ***** Parsing Message: '7' *****
Debug: Detected Microsoft Exchange delivery report type.
Action: delivered
Original-Envelope-Id:
<MBUILDER.1020502110006.25.6538@msgsyd1.emailbill.hpa.com.au>
Reporting-MTA: Exchange;shahingroup.com.au
Arrival-Date: Thu, 2 May 2002 10:39:32 +0930
Original-Recipient: rfc822;gerry@shahingroup.com.au
Status: 2.0.0

Debug: ***** Parsing Message: '8' *****
Debug: Detected Microsoft Exchange delivery report type.

Action: delivered
Original-Envelope-Id:
<MBUILDER.1020502110006.25.6538@msgsyd1.emailbill.hpa.com.au>
Reporting-MTA: Exchange;shahingroup.com.au
Arrival-Date: Thu, 2 May 2002 10:39:32 +0930
Original-Recipient: rfc822;gerry@shahingroup.com.au
Status: 2.0.0

Debug: ***** Parsing Message: '9' *****
Debug: Detected Microsoft Exchange delivery report type.
Action: delivered
Original-Envelope-Id:
<MBUILDER.1020426150007.30.9570@msgsyd1.emailbill.hpa.com.au>
Reporting-MTA: Exchange;shahingroup.com.au
Arrival-Date: Fri, 26 Apr 2002 14:37:35 +0930
Original-Recipient: rfc822;gerry@shahingroup.com.au
Status: 2.0.0

Debug: ***** Parsing Message: '10' *****
Debug: Detected Microsoft Exchange delivery report type.
Action: failed
Original-Envelope-Id:
<MBUILDER.1020421170338.419.23617@msgsyd3.emailbill.hpa.com.au>
Reporting-MTA: Exchange;hpa.com.au
Arrival-Date: Mon, 22 Apr 2002 11:23:45 +1000
Original-Recipient: rfc822;micheal.gerritsen@hpa.com.au
Status: 5.0.0
Diagnostic-Code: smtp;The recipient name is not recognized

Debug: *****
Debug: ***** Parsing Folder: 'pos_match_neg_content' *****

Debug: ***** Parsing Message: '0' *****
Debug: Detected standard MDN report type.
Error: Unable to process MDN report type.

Debug: *****
Debug: ***** Parsing Folder: 'neg_match' *****

Debug: ***** Parsing Message: '0' *****
Error: Unrecognized report type.

Debug: ***** Parsing Message: '1' *****
Error: Unrecognized report type.

Debug: ***** Parsing Message: '2' *****

Error: Unrecognized report type.

Debug: ***** Parsing Message: '3' *****

Error: Unrecognized report type.

Debug: ***** Parsing Message: '4' *****

Error: Unrecognized report type.

Debug: ***** Parsing Message: '5' *****

Error: Unrecognized report type.

Module Name: Builder/apps/mtrk/api/mtrk.h

```
/* MODULE: message_tracking_subsystem_definitions */
```

```
/* COPYRIGHT
```

```
*
```

```
* Copyright (c) MessagingDirect Limited, Edmonton, Canada
```

```
* All rights reserved.
```

```
*
```

```
* Acquisition and use of this software and related materials for any
```

```
* purpose requires a written license agreement from MessagingDirect Limited,
```

```
* or a written license from an organization licensed by MessagingDirect Limited
```

```
* to grant such a license.
```

```
* END COPYRIGHT */
```

```
/* OVERVIEW
```

```
* This module contains the global definitions required for all modules within
```

```
* the M-builder message tracking subsystem.
```

```
* END OVERVIEW */
```

```
#ifndef _message_tracking_subsystem_definitions
```

```
#define _message_tracking_subsystem_definitions
```

```
/* PUBLIC DEPENDENCIES */
```

```
/* END PUBLIC DEPENDENCIES */
```

```
typedef enum {
```

```
    MTRK_FAIL = 1050,
```

```
    MTRK_SUCC = 1051
```

```
} MTRK_RETURN_CODE_t;
```

```
#endif /* _message_tracking_subsystem_definitions */
```

```
/* END MODULE: message_tracking_subsystem_definitions */
```

Module Name: Builder/apps/mtrkon/mtrk_mdn.c

/* MODULE: message_tracking_MDN_parsing */

/* COPYRIGHT

*

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

* All rights reserved.

*

* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,

* or a written license from an organization licensed by MessagingDirect Limited

* to grant such a license.

* END COPYRIGHT */

/* OVERVIEW

* This module contains routines to parse and manage rfc2298 Message Disposition

* Notification content.

* END OVERVIEW */

/* PUBLIC DEPENDENCIES */

#include <stdio.h> /* standard C I/O definitions */

#include <stdlib.h> /* standard C library definitions */

#include <string.h> /* standard C string definitions */

#include <assert.h> /* standard C assertion definitions */

#include "compat.h" /* Compatability library */

#include "eutility.h" /* Extended utility library */

#include "fs.h" /* memory manager API */

#include "gstream.h" /* generic stream API definitions */

#include "mtrk.h" /* message tracking public api */

#include "mtrk_mdn.h" /* message tracking MDN definitions */

/* END PUBLIC DEPENDENCIES */

/* PRIVATE DEPENDENCIES */

/* Private function prototypes */

static MMTR_MDN_INFO_t * /* R: new MDN info structure */

mtrk_mdn_info_new(void);

/* END PRIVATE DEPENDENCIES */

/*

```

*/

/* FUNCTION: MTRK_MDNParse */

/* SYNOPSIS
* Parse the computer parsable part of a Message Disposition Notification (rfc2298).
* END SYNOPSIS */

/* NOTES
* END NOTES */

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if successful,
MTRK_FAIL if failure */
MTRK_MDNParse (
/* PARAMETERS */
    GSTREAM * payload_stream,      /* I: stream to read payload from */
    MMTR_MDN_INFO_t ** mdn_info    /* OA: structure holding parsed MDN
information */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRK_RETURN_CODE_t return_value; /* function return value */
    char * buf;                      /* temporary work buffer */
    char * bp;                       /* buffer pointer */
    char * bp2;                      /* temporary buffer pointer for parsing out lines */
    char * linep;                    /* start of next possible line in buffer */
    char * attr;                     /* DSN attribute field in buffer */
    char * attribute;                /* allocated copy of MDN attribute field */
    char * type;                     /* DSN type field in buffer */
    char * allocated_type;           /* allocated copy of MDN type field */
    char * text;                     /* DSN text field in buffer */
    char * allocated_text;           /* allocated copy of MDN text field */
    int stream_len;                  /* length of payload stream */
    int read_len;                    /* length read from payload stream */
    MMTR_MDN_INFO_t * new_mdn_info; /* MDN information structure to return */
    MDN_DISP_MODE_t disp_mode;       /* disposition mode enumerated type
*/
    MDN_DISP_TYPE_t disp_type;       /* disposition type enumerated type */
/* END VARIABLES */

/* : sanity checks */
    assert (payload_stream != NULL);

/* : initialization */
    return_value = MTRK_SUCC;         /* assumes success */

```

```

stream_len = gstream_size (payload_stream);
buf = (char *) fs_get (stream_len + 1);
linep = buf;

/* : read the payload stream into the work buffer */
read_len = gstream_read (payload_stream, buf, stream_len);
if (read_len != stream_len) {
    cleanup_return (MTRK_FAIL);
}

/* : allocate new MDN information object */
new_mdn_info = mtrk_mdn_info_new();
if (new_mdn_info == NULL) {
    cleanup_return (MTRK_FAIL);
}

/* : walk work buffer to parse each line */
for (bp2 = buf; *bp2 != '\0'; bp2++) {
/* : -- DOC - parse attribute, type and text from line */
    if (*bp2 != '\n') {
        continue;                /* not end of current line */
    }
    *bp2 = '\0';                /* end of current line, \r will be trimmed */
    attr = linep;
    linep = bp2 + 1;            /* start of next possible line */
    bp = strchr (attr, ':'); /* look for attribute token */
    if (bp != NULL) {
        *bp = '\0';
        bp++;
    }
    else {
        bp = attr + strlen(attr);
    }
    type = bp;
    bp = strchr (type, ';'); /* look for possible type token */
    if (bp != NULL) {
        *bp = '\0';
        bp++;
    }
    else {
        bp = type;
        type = NULL;
    }
    text = bp;

/* : - CLAIM: current line is tokenized */

```



```

/* : -- DOC - allocate and trim attribute, type and text tokens */
    attribute = EU_StrDup (attr);
    EU_StrTrim (attribute);
    if (type != NULL) {
        allocated_type = EU_StrDup (type);
        EU_StrTrim (allocated_type);
    }
    allocated_text = EU_StrDup (text);
    EU_StrTrim (allocated_text);

/* : -- DOC - Store type and text into MDN information structure */
    if (strcmp (attribute, "Reporting-UA") == 0 ) {
        if (type == NULL) { /* rfc2298: product is optional, not ua_name
*/
            new_mdn_info->reporting_ua.ua_name = allocated_text;
        }
        else {
            new_mdn_info->reporting_ua.ua_name = allocated_type;
            new_mdn_info->reporting_ua.product = allocated_text;
        }
    }
    else if (strcmp (attribute, "MDN-Gateway") == 0 ) {
        new_mdn_info->mdn_gateway.type = allocated_type;
        new_mdn_info->mdn_gateway.name = allocated_text;
    }
    else if (strcmp (attribute, "Original-Recipient") == 0 ) {
        new_mdn_info->original_recipient.type = allocated_type;
        new_mdn_info->original_recipient.address = allocated_text;
    }
    else if (strcmp (attribute, "Final-Recipient") == 0 ) {
        new_mdn_info->final_recipient.type = allocated_type;
        new_mdn_info->final_recipient.address = allocated_text;
    }
    else if (strcmp (attribute, "Original-Message-Id") == 0 ) {
        new_mdn_info->message_id.id = allocated_text;
    }
    else if (strcmp (attribute, "Failure") == 0 ) {
        new_mdn_info->failure.text = allocated_text;
    }
    else if (strcmp (attribute, "Error") == 0 ) {
        new_mdn_info->error.text = allocated_text;
    }
    else if (strcmp (attribute, "Warning") == 0 ) {
        new_mdn_info->warning.text = allocated_text;
    }
    else if (strcmp (attribute, "Disposition") == 0 ) {

```

```

/* --- DOC - disposition uses 2 enumerated types rather than storing strings */
    if (strncmp (allocated_type, "manual-action/MDN-sent-m", 24) == 0) {
        disp_mode = MDN_DM_MAN_MAN;
    }
    else if (strncmp (allocated_type, "manual-action/MDN-sent-a", 24) == 0) {
        disp_mode = MDN_DM_MAN_AUTO;
    }
    else if (strncmp (allocated_type, "automatic-action/MDN-sent-m", 27) == 0) {
        disp_mode = MDN_DM_AUTO_MAN;
    }
    else if (strncmp (allocated_type, "automatic-action/MDN-sent-a", 27) == 0) {
        disp_mode = MDN_DM_AUTO_AUTO;
    }
    else {
        disp_mode = MDN_DM_UNKNOWN;
    }

    if (strncmp (allocated_text, "failed", 6) == 0) {
        disp_type = MDN_DT_FAILED;
    }
    else if (strncmp (allocated_text, "displayed", 9) == 0) {
        disp_type = MDN_DT_DISPLAYED;
    }
    else if (strncmp (allocated_text, "dispatched", 10) == 0) {
        disp_type = MDN_DT_DISPATCHED;
    }
    else if (strncmp (allocated_text, "processed", 9) == 0) {
        disp_type = MDN_DT_PROCESSED;
    }
    else if (strncmp (allocated_text, "deleted", 7) == 0) {
        disp_type = MDN_DT_DELETED;
    }
    else if (strncmp (allocated_text, "denied", 6) == 0) {
        disp_type = MDN_DT_DENIED;
    }
    else {
        disp_type = MDN_DT_UNKNOWN;
    }

    new_mdn_info->disposition.type = disp_type;
    new_mdn_info->disposition.mode = disp_mode;
    fs_give ((void **) &allocated_type); /* allocated field not stored in structure */
    fs_give ((void **) &allocated_text); /* allocated field not stored in structure */
}
fs_give ((void **) &attribute);

```

```
    }

/* : must return address of pointer to the MDN information structure */
    *mdn_info = new_mdn_info;

/* : cleanup and return */
    CLEANUP:
    if (return_value == MTRK_FAIL) {
        MTRK_MDN_info_destroy (&new_mdn_info);
    }
    fs_give ((void **) &buf);
    return (return_value);
}
/* END FUNCTION: MTRK_MDNParse */

/*
```

```

*/

/* FUNCTION: mtrk_mdn_info_new */

/* SYNOPSIS
   Allocate a new MMTR_MDN_INFO_t structure
  * END SYNOPSIS */

/* NOTES
  * END NOTES */

static MMTR_MDN_INFO_t *          /* R: new MDN info structure */
mtrk_mdn_info_new (
/* PARAMETERS */
/* END PARAMETERS */
)
{
/* VARIABLES */
  MMTR_MDN_INFO_t *new_mdn_info ;/* MDN information structure */
/* END VARIABLES */

/* : sanity checks */

/* : initialization */
  new_mdn_info = NULL;
  new_mdn_info = (MMTR_MDN_INFO_t *) fs_get(sizeof(MMTR_MDN_INFO_t));
  if (new_mdn_info != NULL) {
    memset(new_mdn_info, 0, sizeof(MMTR_MDN_INFO_t));
  }

/* : return new MDN info structure */
  return (new_mdn_info);
}

/* END FUNCTION: mtrk_mdn_info_new */

/*

```

```

*/

/* FUNCTION: MTRK_MDN_info_destroy */

/* SYNOPSIS
* Free all storage for the give MTRK_MDN_INFO_t structure
* END SYNOPSIS */

/* NOTES
* END NOTES */

void                                /* R: no return value */
MTRK_MDN_info_destroy (
/* PARAMETERS */
MMTR_MDN_INFO_t ** mdn_info        /* U: MDN info structure to be destroyed */
/* END PARAMETERS */
)
{

/* : sanity checks */
if (mdn_info == NULL || *mdn_info == NULL) {
    return;
}

/* : walk structure and free fields */
fs_give ((void **) &((*mdn_info)->reporting_ua.ua_name));
fs_give ((void **) &((*mdn_info)->reporting_ua.product));
fs_give ((void **) &((*mdn_info)->mdn_gateway.type));
fs_give ((void **) &((*mdn_info)->mdn_gateway.name));
fs_give ((void **) &((*mdn_info)->original_recipient.type));
fs_give ((void **) &((*mdn_info)->original_recipient.address));
fs_give ((void **) &((*mdn_info)->final_recipient.type));
fs_give ((void **) &((*mdn_info)->final_recipient.address));
fs_give ((void **) &((*mdn_info)->message_id.id));
fs_give ((void **) &((*mdn_info)->failure.text));
fs_give ((void **) &((*mdn_info)->error.text));
fs_give ((void **) &((*mdn_info)->warning.text));

/* free structure */
fs_give ((void **) mdn_info);

/* : return */
return;
}

/* END FUNCTION: MTRK_MDN_info_destroy */

```

/* END MODULE: message_tracking_MDN_parsing */

Module Name: Builder/apps/mtrkmon/mtrkmon.c

/* MODULE: message_monitor_test_harness */

/* COPYRIGHT

*

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

* All rights reserved.

*

* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,

* or a written license from an organization licensed by MessagingDirect Limited

* to grant such a license.

* END COPYRIGHT */

/* OVERVIEW

* This module contains the Message Tracking Monitor (mtrkmon) daemon command function

* definitions. The mtrkmon daemon is an MBuilder Monitor application that monitors

* a set of mailboxes looking for message tracking status messages. Specifically it

* looks for Delivery Status Notifications (DSN) and Message Disposition Notifications

* (MDN). Upon receipt of these messages it, it parses their content and updates

* the Message Tracking Database (mtrkdb) with new status information.

* END OVERVIEW */

/* PUBLIC DEPENDENCIES */

#include <stdio.h> /* standard C I/O definitions */

#include <stdlib.h> /* standard C library definitions */

#include <string.h> /* standard C string definitions */

#include <assert.h> /* standard C assertion definitions */

#include <time.h> /* standard C time definitions */

#include <sys/types.h> /* standard system types */

#include <signal.h> /* OS signal definitions */

#include <unistd.h> /* unix standard functions */

#include "compat.h" /* compatability library */

#include "eutility.h" /* extended utility library */

#include "fs.h" /* memory manager definitions */

#include "mrep.h" /* application runtime message reporting API */

#include "mmonitor.h" /* message monitor API definitions */

#include "mmon_msg.h" /* message monitor message definitions */

#include "mtrk.h" /* message tracking API definitions */

#include "mtrkdb.h" /* message tracking database API definitions */

```

#include "mtrk_dsn.h"                /* message tracking DSN parser definitions
*/
#include "mtrk_mdn.h"                /* message tracking DSN parser definitions
*/
#include "mtrkmon_app.h"              /* message tracking monitor application definitions
*/
/* END PUBLIC DEPENDENCIES */

/* PRIVATE DEPENDENCIES */

/* : DOC - the command environment structure. All of the information used to
: control the operation of the application taken from the command environment
: is held here. Sources of information include the command line arguments,
: and the OS environment. */
typedef struct command_environment {
    char * config_file;               /* pathname of config file */
    BOOLEAN debug_mode;               /* are we in debugging mode */
    BOOLEAN interactive_mode;         /* run in daemon mode or not */
    char * database_dir;              /* message tracking database directory path */
    int n_mailboxes;                  /* number of mailboxes to monitor */
    char ** mailboxes;                 /* list of mailboxes to monitor */
    char *db_path;                    /* path to message tracking database */
} CENV_ENV_t;

/* : DOC - MIME types that we are interested in monitoring in the message tracker. We
: monitor the set of types that are received for DSN and MDN notification in
: message tracking. */
static char * mtrk_mime_types[] = {
    "message/delivery-status",        /* computer parsable part of a DSN */
    "message/disposition-notification", /* computer parsable part of an MDN */
    NULL
};

typedef struct {
    void * db_context;                /* message tracking database context */
    BOOLEAN debug;                     /* whether debugging is on */
} MTRKMON_CB_CONTEXT_t;

/* : private function prototypes */
static MTRK_RETURN_CODE_t cenv_init (int argc, char ** argv, CENV_ENV_t *
cenv);
static MMTR_STATUS MTRK_ProcessReport (void * cb_context, char * mailbox, char
* mime_type, MMTR_SECURITY_VALUES_t * sec_values, GSTREAM *
payload_stream);
static void SetSignalHandlers (void);
static void ShutdownMonitor (int signum);

```


/* END PRIVATE DEPENDENCIES */

/*

```

*/

/* FUNCTION: main */

/* SYNOPSIS
* Main entry point function for message monitor test harness.
* END SYNOPSIS */

/* NOTES
* END NOTES */

int                                /* R: command exit value */
main (
/* PARAMETERS */
    int argc,                      /* I: number of command line arguments */
    char ** argv                   /* I: command line argument vector */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRKDB_CONTEXT_t * db;         /* message tracking database context */
    CENV_ENV_t cenv;               /* command environment info */
    int return_value;              /* function return value */
    MTRKMON_CB_CONTEXT_t cb_context; /* monitor callback context */
/* END VARIABLES */

/* : initialization */
    db = NULL;
    return_value = 0;              /* assume success */

/* : DOC - for policy reasons we will not execute as super user. This is to
   : prevent the routine from being able to inadvertently do anything nasty
   : "outside of its space". */

/* : make sure that we are not doing this as super user */
    if (geteuid() == 0) {
        fprintf(stderr, "You may not run this command as superuser!");
        exit (1);
    }

/* : initialize the command environment */
    if (cenv_init (argc, argv, &cenv) == MTRK_FAIL) {
        exit (1);
    }

/* : if there is not database path then quit */

```

```

    if (cenv.db_path == NULL) {
        fprintf(stderr, "You must specify the database path on the command line or in
configuration.\n");
        cleanup_return (1);
    }
/* : startup the application subsystems */
    if (MTRKMON_AppInit (cenv.db_path,&db) == MTRK_FAIL) {
        fprintf(stderr, "Failed to initialize application subsystems - aborting!\n");
        exit (1);
    }

/* : CLAIM - at this point the application and all dependent subsystems have been
properly
: initialized. */

/* : DOC - loading the runtime configuration will give us information about what
mailboxes
: we are supposed to be monitoring for messages, as well as a few other things. */

/* : load the application runtime configuration */

/* : if there are no mailboxes to monitor then quit */
    if (cenv.n_mailboxes < 1) {
        fprintf(stderr, "You must specify at least one mailbox on the command line or in
configuration.\n");
        cleanup_return (1);
    }

/* : register notification callbacks for MIME types and mailboxes we are interested in */
    debug_trace (DEBUG_CRIT, "Registering notification for mime types ... ");
    cb_context.db_context = db;
    cb_context.debug = cenv.debug_mode;

    MMTR_Register (cenv.mailboxes, mtrk_mime_types, MTRK_ProcessReport, (void *)
&cb_context);
    debug_trace (DEBUG_CRIT, "done.\n");

/* : CLAIM - at this point we have set up the monitor to look at the mailboxes that we
: are expecting to get asynchronous tracking information on. */

/* : setup signal handlers for the command */
    SetSignalHandlers ();

/* : start processing messages arriving in the mailboxes */
    debug_trace (DEBUG_CRIT, "Entering monitoring loop.\n\n");

```

```
MMTR_Start ();
debug_trace (DEBUG_CRIT, "Leaving monitoring loop.\n");

/* : CLAIM - the main processing loop has been interrupted. This is due to a signal
   : or message from the operating system indicating that we should stop processing. */

CLEANUP:

/* : shutdown the dependent subsystems for the application */
MTRKMON_AppShutdown (&db);

/* : cleanup the command environment */

/* : cleanup and return the function result */
return (return_value);
}
/* END FUNCTION: main */

/*
```

```

*/

/* FUNCTION: cenv_init */

/* SYNOPSIS
 * Initialize the command environment for the application. The command environment
 * is created by processing command line options and querying the OS environment.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

static MTRK_RETURN_CODE_t      /* R: MTRK_SUCC if successful,
MTRK_FAIL if failure */
cenv_init (                    /* D: initialize the command environment */
/* PARAMETERS */
    int argc,                  /* I: number of command line arguments */
    char ** argv,              /* I: command line argument vector */
    CENV_ENV_t * cenv          /* U: command environment info */
/* END PARAMETERS */
)
{
/* VARIABLES */
    char opt;                  /* current control argument */
    BOOLEAN cmd_error;         /* error in command line args */
/* END VARIABLES */

/* : sanity checks */
    assert (argv != NULL);
    assert (cenv != NULL);

/* : initialization */
    cmd_error = FALSE;         /* assume command line arguments are OK
*/
    memset ((void *) cenv, '\0', sizeof (CENV_ENV_t));

/* : process control arguments in the command line arguments */
    while (EU_GetOpt (argc, argv, "f:m:p:idx?", &opt) == SUCC) {
        switch (opt) {
            case 'f':
                optarg = EU_GetOptArg ();
                if (optarg == NULL) {
                    fprintf (stderr, "Missing configuration file pathname argument following
\"-f\".\n");
                    cmd_error = TRUE;
                    break;

```

```

    }
    fs_give ((void **) &(cenv->config_file));
    cenv->config_file = EU_StrDup (optarg);
    break;

case 'm':
    optarg = EU_GetOptArg ();
    if (optarg == NULL) {
        fprintf (stderr, "Missing mailbox URL argument following \"-m\".\n");
        cmd_error = TRUE;
        break;
    }
    cenv->n_mailboxes++;
    if (cenv->n_mailboxes == 1) {
        cenv->mailboxes = (char **) fs_get (2 * sizeof(char *));
        cenv->mailboxes[0] = EU_StrDup (optarg);
        cenv->mailboxes[1] = NULL;
    }
    else {
        fs_resize ((void **) &(cenv->mailboxes), ((cenv->n_mailboxes + 1) *
sizeof(char *)));
        cenv->mailboxes[cenv->n_mailboxes - 1] = EU_StrDup (optarg);
        cenv->mailboxes[cenv->n_mailboxes] = NULL;
    }
    break;

case 'p':
    optarg = EU_GetOptArg ();
    if (optarg == NULL) {
        fprintf (stderr, "Missing database pathname argument following \"-
p\".\n");
        cmd_error = TRUE;
        break;
    }
    fs_give ((void **) &(cenv->db_path));
    cenv->db_path = EU_StrDup (optarg);
    break;
case 'd':
    cenv->debug_mode = TRUE;
    break;

case 'i':
    cenv->interactive_mode = TRUE;
    break;

case 'h':

```

```

        case '?':
            cmd_error = TRUE;                /* fake an error to print interface help */
            break;

        default:
            cmd_error = TRUE;
            break;
    }
}

/* : print usage and return failure if there was an error in the command line arguments */
if (cmd_error == TRUE) {
    printf ("usage: %s [-d] [-i] [-f config_file_path] [-m mailbox_url] [-p
database_path]\n", argv[0]);
    return (MTRK_FAIL);
}

/* : cleanup and return */
return (MTRK_SUCC);
}
/* END FUNCTION: cenv_init */

/*

```

```

*/

/* FUNCTION: MTRK_ProcessReport */

/* SYNOPSIS
 * Message notification callback function for handling message tracking content.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

static MMTR_STATUS                                /* R: MMTR_SUCC if successful,
MMTR_FAIL if failure */
MTRK_ProcessReport (
/* PARAMETERS */
    void * cb_context,                          /* I: application context data for callback */
    char * mailbox,                             /* I: mailbox notification occurred on */
    char * mime_type,                           /* I: MIME type of received message */
    MMTR_SECURITY_VALUES_t * sec_values,        /* I: security values associated with
received message */
    GSTREAM * payload_stream                    /* I: open data stream to decoded MIME
payload */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MMTR_DSN_INFO_t * dsn_info;                 /* parsed DSN data from payload */
    MMTR_MDN_INFO_t * mdn_info;                 /* parsed MDN data from payload */
    MTRK_STATUS_REC_t status_rec;               /* message tracking status record */
    MTRKDB_CONTEXT_t * db;                     /* message tracking database context */
    MMTR_STATUS return_value;                   /* function return value */
    time_t cur_time;                           /* current time (raw) */
    BOOLEAN debug;                             /* whether debug logging is enabled */
/* END VARIABLES */

/* : sanity checks */
    if (cb_context == NULL) {
        return (MMTR_FAIL);
    }
    if (mailbox == NULL) {
        return (MMTR_FAIL);
    }
    if (mime_type == NULL) {
        return (MMTR_FAIL);
    }
    if (sec_values == NULL) {

```



```

        return (MMTR_FAIL);
    }
    if (payload_stream == NULL) {
        return (MMTR_FAIL);
    }

/* : initialization */
    dsn_info = NULL;
    mdn_info = NULL;
    memset ((void *) &status_rec, '\0', sizeof (MTRK_STATUS_REC_t));

/* monitor callback context contains the database handle */
    db = (MTRKDB_CONTEXT_t*)((MTRKMON_CB_CONTEXT_t *) cb_context)-
>db_context;
    debug = ((MTRKMON_CB_CONTEXT_t *) cb_context)->debug;
    return_value = MMTR_SUCC;          /* assume success */

/* : print some telemetry */
    debug_trace (DEBUG_MED, "Payload from mailbox: %s\n", mailbox);
    debug_trace (DEBUG_MED, "Payload MIME type: %s\n", mime_type);

/* : parse a DSN payload into a message tracking status record */
    if (strcmp (mtrk_mime_types[0], mime_type) == 0) {
        if (MTRK_DSNParse (payload_stream, &dsn_info) == MTRK_FAIL) {
            cleanup_return (MMTR_FAIL);
        }

        status_rec.msgid = dsn_info->envelope_id.id;          /* unique
message identifier */
        status_rec.host = dsn_info->reporting_mta.name;      /* host to which status
record applies */
        status_rec.to_host = dsn_info->dsn_gateway.name;
        if (dsn_info->original_recipient.address != NULL) {
            status_rec.user = dsn_info->original_recipient.address;
        }
        else {
            status_rec.user = dsn_info->final_recipient.address;
        }
        status_rec.timestamp = time(&cur_time);              /* creation timestamp
for status record */

        switch (dsn_info->mta_action.action) {

/* : -- message cannot be delivered to destination mailbox */
        case DSN_AC_FAILED:

```

```

        status_rec.status = MTRK_SC_REJECTED;          /* status of message
*/
                                /* status reason description (optional) */
        status_rec.description = dsn_info->diagnostic_code.text;
        break;

/* : -- message has been temporarily delayed in transit at this host */
    case DSN_AC_DELAYED:
        status_rec.status = MTRK_SC_DELAYED;          /* status of message
*/
                                /* status reason description (optional) */
        status_rec.description = dsn_info->diagnostic_code.text;
        break;

/* : -- message has been delivered to final destination mailbox */
    case DSN_AC_DELIVERED:
        status_rec.status = MTRK_SC_DELIVERED;        /* status of message
*/
        break;

/* : -- message has been relayed to a non-DSN capable host */
    case DSN_AC_RELAYED:
        status_rec.status = MTRK_SC_BLACKHOLE;        /* status of message
*/
                                /* status reason description (optional) */
        status_rec.description = dsn_info->diagnostic_code.text;
        break;

/* : -- message has been delivered and forwarded after expansion */
    case DSN_AC_EXPANDED:
        status_rec.status = MTRK_SC_FORWARDED;        /* status of message
*/
        break;
    }
}

/* : otherwise parse an MDN payload into a message tracking status record */
else if (strcmp (mtrk_mime_types[1], mime_type) == 0) {
    if (MTRK_MDNParse (payload_stream, &mdn_info) == MTRK_FAIL) {
        cleanup_return (MMTR_FAIL);
    }

    status_rec.msgid = mdn_info->message_id.id;        /* unique
message identifier */
    status_rec.host = mdn_info->reporting_ua.ua_name;  /* host to which
status record applies */

```

```

        status_rec.to_host = mdn_info->mdn_gateway.name;
        if (mdn_info->original_recipient.address != NULL) {
            status_rec.user = mdn_info->original_recipient.address;
        }
        else {
            status_rec.user = mdn_info->final_recipient.address;
        }
        status_rec.timestamp = time(&cur_time);           /* creation timestamp
for status record */

        switch (mdn_info->disposition.type) {

/* : -- proper MDN not generated due to failure. See Failure field */
        case MDN_DT_FAILED:
            status_rec.status = MTRK_SC_REJECTED;         /* status of message
*/
            status_rec.description = mdn_info->failure.text; /* status reason description
(optional) */
            break;

/* : -- message has been displayed */
/* : -- message has been sent on (e.g. printed/faxed/forwarded) */
/* : -- message processed without displaying */
/* : -- message has been deleted */
/* : -- recipient did not allow MDN to be sent */
        case MDN_DT_DISPLAYED:
        case MDN_DT_DISPACHED:
        case MDN_DT_PROCESSED:
        case MDN_DT_DELETED:
        case MDN_DT_DENIED:
            status_rec.status = MTRK_SC_SEEN;             /* status of message */
            break;
        }
    }

/* : otherwise it is a bogus MIME type that we shouldn't have been notified on */
    else {
        debug_trace (DEBUG_CRIT, "Recieved notification for MIME type that was not
registered: %s\n", mime_type);
        cleanup_return (MMTR_FAIL);
    }

/* : CLAIM - at this point we have translated the report into a status record. */

/* : add the record to the message tracking database based on the report */
    if (MTRKDB_RecAdd (db, &status_rec) == MTRK_FAIL) {

```

```

        cleanup_return (MMTR_FAIL);
    }

/* : log the message id */
    if (debug == TRUE) {
        debug_trace(DEBUG_CRIT,"Saved message with msgid %s, status %d, user
%s\n",
                    status_rec.msgid,status_rec.status,status_rec.user);
    }

/* : cleanup and return the function return value */
CLEANUP:
    if (dsn_info != NULL) {
        MTRK_DSN_info_destroy (&dsn_info);
    }
    if (mdn_info != NULL) {
        MTRK_MDN_info_destroy (&mdn_info);
    }
    return (return_value);
}
/* END FUNCTION: MTRK_ProcessReport */

/*

```

```

*/

/* FUNCTION: SetSignalHandlers */

/* SYNOPSIS
 * Setup signal handlers for the command.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

static void                                /* R: no return value */
SetSignalHandlers (
/* PARAMETERS */
    void
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : setup shutdown handler for shutdown signals */
    signal(SIGHUP, ShutdownMonitor);
    signal(SIGINT, ShutdownMonitor);
    signal(SIGQUIT, ShutdownMonitor);
    signal(SIGTERM, ShutdownMonitor);

/* : setup ignore handlers for the signals we want to ignore */
    signal(SIGPIPE, SIG_IGN);

/* : cleanup and return */
    return;
}
/* END FUNCTION: SetSignalHandlers */

/*

```

```

*/

/* FUNCTION: ShutdownMonitor */

/* SYNOPSIS
* Handle a signal by shutting down the daemon application. We try to clean
* up and make sure that the system is in a consistent state before we exit
* the process.
* END SYNOPSIS */

/* NOTES
* END NOTES */

static void                               /* R: no return value */
ShutdownMonitor (
/* PARAMETERS */
    int signum                           /* I: number of received signal */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : log the receipt of the signal */
    debug_trace (DEBUG_CRIT, "Caught signal %d.\n", signum);

/* : DOC - we wish to signal the application that we are done processing. We
    : do this by telling the mail monitor subsystem to stop processing mailboxes. */

/* : tell the mail monitor to stop */
    MMTR_Stop ();

/* : cleanup and return */
    return;
}
/* END FUNCTION: ShutdownMonitor */

/* END MODULE: message_monitor_test_harness */

```

Module Name: Builder/monitor/apps/mtrkmon/mtrk_dsn.c

/* MODULE: message_tracking_DSN_parsing */

/* COPYRIGHT

*

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

* All rights reserved.

*

* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,

* or a written license from an organization licensed by MessagingDirect Limited

* to grant such a license.

* END COPYRIGHT */

/* OVERVIEW

* This module contains routines to parse and manage rfc1894 Delivery Status

* Notification content.

* END OVERVIEW */

/* PUBLIC DEPENDENCIES */

#include <stdio.h> /* standard C I/O definitions */

#include <stdlib.h> /* standard C library definitions */

#include <string.h> /* standard C string definitions */

#include <assert.h> /* standard C assertion definitions */

#include "compat.h" /* Compatability library */

#include "eutility.h" /* Extended utility library */

#include "fs.h" /* memory manager API */

#include "gstream.h" /* generic stream API definitions */

#include "mtrk.h" /* message tracking public api */

#include "mtrk_dsn.h" /* message tracking DSN definitions */

/* END PUBLIC DEPENDENCIES */

/* PRIVATE DEPENDENCIES */

static MMTR_DSN_INFO_t * /* R: new DSN info structure */

mtrk_dsn_info_new(void);

/* END PRIVATE DEPENDENCIES */

/*

```

*/

/* FUNCTION: MTRK_DSNParse */

/* SYNOPSIS
 * Parse the computer parsable part of a Delivery Status Notification (rfc1894).
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if successful,
MTRK_FAIL if failure */
MTRK_DSNParse (
/* PARAMETERS */
    GSTREAM * payload_stream,      /* I: stream to read payload from */
    MMTR_DSN_INFO_t ** dsn_info    /* OA: structure holding parsed DSN
information */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRK_RETURN_CODE_t return_value; /* function return value */
    char * buf;                      /* temporary work buffer */
    char * bp;                       /* buffer pointer */
    char * bp2;                      /* temporary buffer pointer for parsing out lines */
    char * linep;                    /* start of next possible line in buffer */
    char * attr;                     /* DSN attribute field in buffer */
    char * attribute;                /* allocated copy of DSN attribute field */
    char * type;                     /* DSN type field in buffer */
    char * all_type;                 /* allocated copy of DSN type field */
    char * text;                     /* DSN text field in buffer */
    char * all_text;                 /* allocated copy of DSN text field */
    int stream_len;                  /* length of payload stream */
    int read_len;                    /* length read from payload stream */
    MMTR_DSN_INFO_t * new_dsn_info; /* DSN information structure to return */
    DSN_ACTION_CODE_t action;        /* Action code enumerated type */

/* END VARIABLES */

/* : sanity checks */
    assert (payload_stream != NULL);

/* : initialization */
    return_value = MTRK_SUCC;        /* assumes success */
    stream_len = gstream_size (payload_stream);

```



```

buf = (char *) fs_get (stream_len + 1);
linep = buf;

/* : read the payload stream into the work buffer */
read_len = gstream_read (payload_stream, buf, stream_len);
if (read_len != stream_len) {
    cleanup_return (MTRK_FAIL);
}

/* : allocate new DSN information object */
new_dsn_info = mtrk_dsn_info_new( );
if (new_dsn_info == NULL) {
    cleanup_return (MTRK_FAIL);
}

/* : walk work buffer to parse each line */
for (bp2 = buf; *bp2 != '\0'; bp2++) {
/* : parse each line */
    if (*bp2 != '\n') {
        continue;                /* not end of current line */
    }
    *bp2 = '\0';                /* end of current line, \r will be trimmed */
    attr = linep;
    linep = bp2 + 1;            /* start of next possible line */
    bp = strchr (attr, ':'); /* look for attribute token */
    if (bp != NULL) {
        *bp = '\0';
        bp++;
    }
    else {
        bp = attr + strlen(attr);
    }
    type = bp;
    bp = strchr (type, ';'); /* look for possible type token */
    if (bp != NULL) {
        *bp = '\0';
        bp++;
    }
    else {
        bp = type;
        type = NULL;
    }
    text = bp;

/* : - CLAIM: current line is tokenized */
    attribute = EU_StrDup (attr);

```

```

EU_StrTrim (attribute);
if (type != NULL) {
    all_type = EU_StrDup (type);
    EU_StrTrim (all_type);
}
all_text = EU_StrDup (text);
EU_StrTrim (all_text);

if (strcmp (attribute, "Original-Envelope-Id") == 0 ) {
    new_dsn_info->envelope_id.id = all_text;
}
else if (strcmp (attribute, "Reporting-MTA") == 0 ) {
    new_dsn_info->reporting_mta.type = all_type;
    new_dsn_info->reporting_mta.name = all_text;
}
else if (strcmp (attribute, "DSN-Gateway") == 0 ) {
    new_dsn_info->dsn_gateway.type = all_type;
    new_dsn_info->dsn_gateway.name = all_text;
}
else if (strcmp (attribute, "Received-From-MTA") == 0 ) {
    new_dsn_info->from_mta.type = all_type;
    new_dsn_info->from_mta.name = all_text;
}
else if (strcmp (attribute, "Arrival-Date") == 0 ) {
    new_dsn_info->arrival_date.date = all_text;
}
else if (strcmp (attribute, "Original-Recipient") == 0 ) {
    new_dsn_info->original_recipient.type = all_type;
    new_dsn_info->original_recipient.address = all_text;
}
else if (strcmp (attribute, "Final-Recipient") == 0 ) {
    new_dsn_info->final_recipient.type = all_type;
    new_dsn_info->final_recipient.address = all_text;
}
else if (strcmp (attribute, "Diagnostic-Code") == 0 ) {
    new_dsn_info->diagnostic_code.type = all_type;
    new_dsn_info->diagnostic_code.text = all_text;
}
else if (strcmp (attribute, "Action") == 0 ) {
    if (strcmp (all_text, "failed", 6) == 0) {
        action = DSN_AC_FAILED;
    }
    else if (strcmp (all_text, "delayed", 7) == 0) {
        action = DSN_AC_DELAYED;
    }
    else if (strcmp (all_text, "delivered", 9) == 0) {

```

```

        action = DSN_AC_DELIVERED;
    }
    else if (strncmp (all_text, "relayed", 7) == 0) {
        action = DSN_AC_RELAYED;
    }
    else if (strncmp (all_text, "expanded", 8) == 0) {
        action = DSN_AC_EXPANDED;
    }
    else {
        action = DSN_AC_UNKNOWN;
    }
    new_dsn_info->mta_action.action = action ;
    fs_give ((void **) &all_text); /* allocated field not stored in structure */
}
else if (strcmp (attribute, "Status") == 0 ) {
    new_dsn_info->delivery_status.status = all_text;
}
else if (strcmp (attribute, "Remote-MTA") == 0 ) {
    new_dsn_info->remote_mta.type = all_type;
    new_dsn_info->remote_mta.name = all_text;
}
else if (strcmp (attribute, "Diagnostic-Code") == 0 ) {
    new_dsn_info->diagnostic_code.type = all_type;
    new_dsn_info->diagnostic_code.text = all_text;
}
else if (strcmp (attribute, "Last-Attempt-Date") == 0 ) {
    new_dsn_info->last_attempt.date = all_text;
}
else if (strcmp (attribute, "Will-Retry-Until") == 0 ) {
    new_dsn_info->try_until.date = all_text;
}
fs_give ((void **) &attribute);

}

/* : must return address of pointer to the DSN information structure */
*dsn_info = new_dsn_info;

/* : cleanup and return success */
CLEANUP:
    if (return_value == MTRK_FAIL) {
        MTRK_DSN_info_destroy (&new_dsn_info);
    }
    fs_give ((void **) &buf);
    return (return_value);
}

```

```
/* END FUNCTION: MTRK_DSNParse */
```

```
/*
```

```

*/

/* FUNCTION: mtrk_dsn_info_new */

/* SYNOPSIS
   Allocate a new MMTR_DSN_INFO_t structure
  * END SYNOPSIS */

/* NOTES
  * END NOTES */

static MMTR_DSN_INFO_t *          /* R: new DSN info structure */
mtrk_dsn_info_new (
/* PARAMETERS */
/* END PARAMETERS */
)
{
/* VARIABLES */
  MMTR_DSN_INFO_t *new_dsn_info ; /* DSN information structure */
/* END VARIABLES */

/* : sanity checks */

/* : initialization */
  new_dsn_info = NULL;
  new_dsn_info = (MMTR_DSN_INFO_t *) fs_get(sizeof(MMTR_DSN_INFO_t));
  if (new_dsn_info != NULL) {
    memset(new_dsn_info, 0, sizeof(MMTR_DSN_INFO_t));
  }

/* : return new DSN info structure */
  return (new_dsn_info);
}

/* END FUNCTION: mtrk_dsn_info_new */

/*

```

```

*/

/* FUNCTION: MTRK_DSN_info_destroy */

/* SYNOPSIS
* Free all storage for the give MTRK_DSN_INFO_t structure
* END SYNOPSIS */

/* NOTES
* END NOTES */

void                                /* R: no return value */
MTRK_DSN_info_destroy (
/* PARAMETERS */
MMTR_DSN_INFO_t ** dsn_info        /* U: DSN info structure to be destroyed */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
if (dsn_info == NULL || *dsn_info == NULL) {
    return;
}

/* : walk structure and free fields */
fs_give ((void **) &((*dsn_info)->envelope_id.id));
fs_give ((void **) &((*dsn_info)->reporting_mta.type));
fs_give ((void **) &((*dsn_info)->reporting_mta.name));
fs_give ((void **) &((*dsn_info)->dsn_gateway.type));
fs_give ((void **) &((*dsn_info)->dsn_gateway.name));
fs_give ((void **) &((*dsn_info)->from_mta.type));
fs_give ((void **) &((*dsn_info)->from_mta.name));
fs_give ((void **) &((*dsn_info)->arrival_date.date));
fs_give ((void **) &((*dsn_info)->original_recipient.type));
fs_give ((void **) &((*dsn_info)->original_recipient.address));
fs_give ((void **) &((*dsn_info)->final_recipient.type));
fs_give ((void **) &((*dsn_info)->final_recipient.address));
fs_give ((void **) &((*dsn_info)->delivery_status.status));
fs_give ((void **) &((*dsn_info)->remote_mta.type));
fs_give ((void **) &((*dsn_info)->remote_mta.name));
fs_give ((void **) &((*dsn_info)->diagnostic_code.type));
fs_give ((void **) &((*dsn_info)->diagnostic_code.text));
fs_give ((void **) &((*dsn_info)->last_attempt.date));
fs_give ((void **) &((*dsn_info)->try_until.date));

```

```
/* free structure */
    fs_give ((void **) dsn_info);

/* : return */
    return;
}

/* END FUNCTION: MTRK_DSN_info_destroy */

/* END MODULE: message_tracking_DSN_parsing */
```

Module Name: Builder/monitor/mmonitor.c

/* MODULE: message_monitor_API */

/* COPYRIGHT

*

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

* All rights reserved.

*

* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,

* or a written license from an organization licensed by MessagingDirect Limited

* to grant such a license.

* END COPYRIGHT */

/* OVERVIEW

* This module contains definitions for the functions (methods) for the message

* monitor API.

* END OVERVIEW */

/* PUBLIC DEPENDENCIES */

#include <stdio.h> /* standard C I/O definitions */

#include <stdlib.h> /* standard C library definitions */

#include <string.h> /* standard C string definitions */

#include <assert.h> /* standard C assertion definitions */

#include <unistd.h> /* unix standard functions */

#include "compat.h" /* Compatability library */

#include "eutility.h" /* Extended utility library */

#include "fs.h" /* memory manager definitions */

#include "mrep.h" /* application runtime message reporting definitions

*/

#include "mmmonitor.h" /* message monitor API definitions */

#include "mmmonitor_int.h" /* mail monitor private definitions */

#include "mmon_msg.h" /* mail monitor runtime messages */

/* END PUBLIC DEPENDENCIES */

/* PRIVATE DEPENDENCIES */

/* : DOC - Mailbox monitor notification list. This is a list of mailbox notification
: entries that defines the set of mailboxes that we should monitor. */

LISTPTR g_mailbox_list = NULL;

/* : DOC - monitor processing state. This controls the steady state processing
: of the monitor. */


```

enum {
    MMTR_STATE_START,          /* initial state */
    MMTR_STATE_MONITOR,        /* monitor mailboxes for incoming
messages */
    MMTR_STATE_STOP            /* finished monitoring mailboxes */
} g_process_state;

/* : private function prototypes */
static MMTR_MLIST_ENTRY_t * MMTR_MailboxEntryNew (char * mailbox_url);
static void                MMTR_MailboxEntryDestroy (MMTR_MLIST_ENTRY_t **
entry);
static MMTR_MLIST_ENTRY_t * MMTR_MailboxEntryGet (char * mailbox_url);
static MMTR_MIME_REG_t *   MMTR_MIMERegNew (char * mime_type);
static void                MMTR_MIMERegDestroy (MMTR_MIME_REG_t ** mime_reg);
static MMTR_MIME_REG_t *   MMTR_MIMERegGet (MMTR_MLIST_ENTRY_t *
entry, char * mime_type);

/* END PRIVATE DEPENDENCIES */

/*

```

```

*/

/* FUNCTION: MMTR_Init */

/* SYNOPSIS
* Initialize the message monitor subsystem.
* END SYNOPSIS */

/* NOTES
* END NOTES */

MMTR_STATUS                                /* R: MMTR_SUCC if successful,
MMTR_FAIL if failure */
MMTR_Init (
/* PARAMETERS */
void
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
assert (g_mailbox_list == NULL); /* must NOT have already initialized */

/* : DOC - the "msg" subsystem better have been initialized prior to calling this. We
: presume that the mbuilder MB_Init routine has been called prior to the monitor
: subsystem being initialized. */

/* : load the message monitor runtime message table */
if (MSG_add_msg_table (&MMON_mt_info) == FAIL) {
    debug_trace (DEBUG_CRIT, "Failed to load the mandatory message table \"%s\"
from the file \"%s\".\n",
                MMON_mt_info.table_name,
                MMON_mt_info.filename);
    return (MMTR_FAIL);
}

/* : initialize the mailbox monitor list */
rc_ll_list_new (&(g_mailbox_list));

/* : cleanup and return success */
return (MMTR_SUCC);
}
/* END FUNCTION: MMTR_Init */

```

/*

```

*/

/* FUNCTION: MMTR_Shutdown */

/* SYNOPSIS
 * Shutdown the message monitor subsystem.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

void                                /* R: no return value */
MMTR_Shutdown (
/* PARAMETERS */
void
/* END PARAMETERS */
)
{
/* VARIABLES */
    CELLPTR curcell;                /* current cell in handle list */
    MMTR_MLIST_ENTRY_t * entry;      /* mailbox list entry */
/* END VARIABLES */

/* : sanity checks */
    if (g_mailbox_list == NULL) {
        return;
    }

/* : walk the mailbox list closing and destroying monitor entries */
    for (curcell = rc_ll_cell_first (g_mailbox_list);
         curcell != NULL;
         curcell = rc_ll_cell_next (curcell)) {

/* : - get the mailbox list entry from the cell */
        entry = (MMTR_MLIST_ENTRY_t *) rc_ll_cell_value (curcell);

/* : - destroy the cache entry */
        MMTR_MailboxEntryDestroy (&entry);
    }

/* : destroy the mailbox monitor list */
    rc_ll_list_destroy (&(g_mailbox_list), FALSE);

/* : cleanup and return */
    return;
}

```

```
/* END FUNCTION: MMTR_Shutdown */
```

```
/*
```

```

*/

/* FUNCTION: MMTR_Start */

/* SYNOPSIS
* Start the message monitor. The message monitor is a steady state
* loop that will not return unless it has received a platform dependent
* termination event. Termination events are initiated by the MMTR_Stop
* function.
* END SYNOPSIS */

/* NOTES
* END NOTES */

MMTR_STATUS                                /* R: MMTR_SUCC if successful,
MMTR_FAIL if failure */
MMTR_Start (
/* PARAMETERS */
void
/* END PARAMETERS */
)
{
/* VARIABLES */
CELLPTR curcell;                          /* current cell in handle list */
MMTR_MLIST_ENTRY_t * entry;               /* mailbox list entry */
/* END VARIABLES */

/* : sanity checks */
assert (g_mailbox_list != NULL);

/* : move into PROCESS state */
g_process_state = MMTR_STATE_MONITOR;

/* : loop while we are in PROCESS state */
while (g_process_state == MMTR_STATE_MONITOR) {

/* : - walk the mailbox list checking for new messages */
for (curcell = rc_ll_cell_first (g_mailbox_list);
    curcell != NULL;
    curcell = rc_ll_cell_next (curcell)) {

/* : -- get the mailbox list entry from the cell */
entry = (MMTR_MLIST_ENTRY_t *) rc_ll_cell_value (curcell);

/* : -- process any new messages in the mailbox */
MMTR_ProcessMailboxEntry (entry);

```

```
    }

/* : - sleep for a bit before we go bashing the server again */
    sleep(5);
}

/* : cleanup and return success */
return (MMTR_SUCC);
}
/* END FUNCTION: MMTR_Start */

/*
```

```

*/

/* FUNCTION: MMTR_Stop */

/* SYNOPSIS
 * Stop the message monitor.
 * END SYNOPSIS */

/* NOTES
 * Stopping the message monitor is simple. We set a global state variable that
 * signals all steady state loop instances of the message monitor that they should
 * exit from their loops and return.
 * END NOTES */

void                                /* R: no return value */
MMTR_Stop (
/* PARAMETERS */
void
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : move into STOP state */
g_process_state = MMTR_STATE_STOP;

/* : cleanup and return */
return;
}
/* END FUNCTION: MMTR_Stop */

/*

```



```

*/

/* FUNCTION: MMTR_Register */

/* SYNOPSIS
* Register a notification for MIME types and mailboxes. Notifications are
* delivered via a callback function. This routine registers a single callback
* for a group of mailboxes and MIME types.
*
* Mailboxes are specified as an array of mailbox URL strings, with a
* terminating NULL string pointer. Currently supported mailbox URL
* schemes include:
*
* imap      - remote mailbox accessed by IMAP4 protocol
* pop       - remote mailbox accessed by POP3 protocol
* folder    - local (disk based) mailbox
*
* Callers MUST provide at least one mailbox to monitor.
*
* MIME types are specified as an array of MIME type strings of the form
"major/minor",
* with a terminating NULL string pointer.
* END SYNOPSIS */

/* NOTES
* END NOTES */

MMTR_STATUS                                /* R: MMTR_SUCC if successful,
MMTR_FAIL if failure */
MMTR_Register (
/* PARAMETERS */
char ** mailboxes,                        /* I: list of mailbox URL's to monitor */
char ** mime_types,                      /* I: list of mime types we want notification for */
MMTR_ReceiptNotificationCB_ft notify_cb, /* I: notification callback function */
void * cb_context                        /* I: application context data for callback function */
)
{
/* VARIABLES */
int i;                                  /* loop counter */
int j;                                  /* loop counter */
MMTR_MLIST_ENTRY_t * entry;             /* mailbox notification entry */
MMTR_MIME_REG_t * mime_reg;             /* mime type registration object */
MMTR_NOT_CB_ENTRY_t * cb_entry; /* mime type registration callback entry */
/* END VARIABLES */

```

```

/* : sanity checks */
assert (mailboxes != NULL);
assert (mailboxes[0] != NULL); /* MUST provide at least one mailbox to monitor */
assert (mime_types != NULL);
assert (mime_types[0] != NULL); /* MUST provide at least one MIME type to
monitor */
assert (notify_cb != NULL);

/* : initialization */

/* : walk the list of mailboxes adding/updating mailbox notification entries */
for (i = 0; mailboxes[i] != NULL; i++) {

/* : - if there is no notification entry for the mailbox then create one */
entry = MMTR_MailboxEntryGet (mailboxes[i]);
if (entry == NULL) {
entry = MMTR_MailboxEntryNew (mailboxes[i]);
if (entry == NULL) {
return (MMTR_FAIL);
}
rc_ll_cell_append (g_mailbox_list, (CVALPTR) entry);
}

/* : - walk the list of MIME type registrations adding/updating callback lists */
for (j = 0; mime_types[j] != NULL; j++) {

/* : -- if there is no registration for the MIME type then create one */
mime_reg = MMTR_MIMERegGet (entry, mime_types[j]);
if (mime_reg == NULL) {
mime_reg = MMTR_MIMERegNew (mime_types[j]);
rc_ll_cell_append (entry->mime_notifiers, (CVALPTR) mime_reg);
}

/* : -- add a new callback entry to the MIME type registration */
cb_entry = (MMTR_NOT_CB_ENTRY_t *) fs_get
(sizeof(MMTR_NOT_CB_ENTRY_t));
cb_entry->cb_context = cb_context;
cb_entry->cb_func = notify_cb;
rc_ll_cell_append (mime_reg->cb_list, (CVALPTR) cb_entry);
}
}

/* : cleanup and return success */
return (MMTR_SUCC);
}

```

```
/* END FUNCTION: MMTR_Register */
```

```
/*
```

```

*/

/* FUNCTION: MMTR_MailboxEntryNew */

/* SYNOPSIS
 * Construct a new mailbox notification entry object.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

static MMTR_MLIST_ENTRY_t *          /* R: new mailbox notification entry */
MMTR_MailboxEntryNew (
/* PARAMETERS */
    char * mailbox_url                /* I: mailbox to create entry for */
/* END PARAMETERS */
)
{
/* VARIABLES */
    FOLDER fid;                       /* open folder handle */
    MMTR_MLIST_ENTRY_t * entry;       /* new entry to return */
/* END VARIABLES */

/* : sanity checks */
    assert (mailbox_url != NULL);

/* : open a connection to the mailbox */
    fid = MM_FolOpenURL (mailbox_url, FALSE);
    if (fid == INVALID_FOLDER) {
        return (NULL);
    }

/* : allocate and initialize a new mailbox entry */
    entry = (MMTR_MLIST_ENTRY_t *) fs_get (sizeof(MMTR_MLIST_ENTRY_t));
    entry->fid = fid;
    entry->mailbox_url = EU_StrDup (mailbox_url);
    rc_ll_list_new (&(entry->mime_notifiers));

/* : cleanup and return the new entry */
    return (entry);
}
/* END FUNCTION: MMTR_MailboxEntryNew */

/*

```

```

*/

/* FUNCTION: MMTR_MailboxEntryDestroy */

/* SYNOPSIS
* Destroy a mailbox notification entry object.
* END SYNOPSIS */

/* NOTES
* END NOTES */

static void                                /* R: no return value */
MMTR_MailboxEntryDestroy (
/* PARAMETERS */
    MMTR_MLIST_ENTRY_t ** entry          /* U: entry to destroy */
/* END PARAMETERS */
)
{
/* VARIABLES */
    CELLPTR curcell;                     /* current cell in handle list */
    MMTR_MIME_REG_t * mime_reg;          /* MIME registration record */
/* END VARIABLES */

/* : sanity checks */
    assert (entry != NULL);
    if (*entry == NULL) {
        return;
    }

/* : close the folder stream */
    if ((*entry)->fid != INVALID_FOLDER) {
        MM_FolClose ((*entry)->fid, FALSE);
    }

/* : destroy the folder path URL */
    fs_give ((void **) &(*entry)->mailbox_url);

/* : walk the mailbox list closing and destroying monitor entries */
    for (curcell = rc_ll_cell_first ((*entry)->mime_notifiers);
        curcell != NULL;
        curcell = rc_ll_cell_next (curcell)) {

/* : - get the mime registration from the registration list */
        mime_reg = (MMTR_MIME_REG_t *) rc_ll_cell_value (curcell);

/* : - destroy the mime registration record */

```

```
        MMTR_MIMERegDestroy (&(mime_reg));
    }

    /* : destroy the mailbox list */
    rc_ll_list_destroy (&((*entry)->mime_notifiers), FALSE);

    /* : destroy the mailbox entry */
    fs_give ((void **) entry);

    /* : cleanup and return */
    return;
}
/* END FUNCTION: MMTR_MailboxEntryDestroy */

/*
```

```

*/

/* FUNCTION: MMTR_MailboxEntryGet */

/* SYNOPSIS
 * Lookup and return the mailbox notification entry that matches a mailbox
 * URL.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

static MMTR_MLIST_ENTRY_t *          /* R: mailbox notification entry matching
mailbox URL */
MMTR_MailboxEntryGet (
/* PARAMETERS */
char * mailbox_url                    /* I: mailbox URL to get entry for */
/* END PARAMETERS */
)
{
/* VARIABLES */
CELLPTR curcell;                      /* current cell in handle list */
MMTR_MLIST_ENTRY_t * entry;           /* mailbox list entry to return */
/* END VARIABLES */

/* : sanity checks */
assert (mailbox_url != NULL);

/* : walk the mailbox list matching monitor entries */
for (curcell = rc_ll_cell_first (g_mailbox_list);
    curcell != NULL;
    curcell = rc_ll_cell_next (curcell)) {

/* : - get the mailbox list entry from the cell */
    entry = (MMTR_MLIST_ENTRY_t *) rc_ll_cell_value (curcell);

/* : - try to match the mailbox entry */
    if (strcmp (entry->mailbox_url, mailbox_url) == 0) {
        return (entry);
    }
}

/* : CLAIM - if we get here then we did not find the mailbox notification entry
: that we were looking for. */

/* : cleanup and return NULL */

```

```
    return (NULL);  
}  
/* END FUNCTION: MMTR_MailboxEntryGet */  
  
/*
```



```

*/

/* FUNCTION: MMTR_MIMERegNew */

/* SYNOPSIS
* Construct and return a new MIME registration object for a particular MIME type.
* END SYNOPSIS */

/* NOTES
* END NOTES */

static MMTR_MIME_REG_t *          /* R: new MIME registration object */
MMTR_MIMERegNew (
/* PARAMETERS */
    char * mime_type              /* I: MIME type to do registration for */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MMTR_MIME_REG_t * mime_reg;    /* new MIME registration object to
return */
/* END VARIABLES */

/* : sanity checks */
    assert (mime_type != NULL);

/* : allocate and initialize a new MIME registration object */
    mime_reg = (MMTR_MIME_REG_t *) fs_get (sizeof(MMTR_MIME_REG_t));
    mime_reg->mime_type = EU_StrDup (mime_type);
    rc_ll_list_new (&(mime_reg->cb_list));

/* : cleanup and return the new registration object */
    return (mime_reg);
}
/* END FUNCTION: MMTR_MIMERegNew */

/*

```

```

*/

/* FUNCTION: MMTR_MIMERegDestroy */

/* SYNOPSIS
* Destroy a MIME notification registration object.
* END SYNOPSIS */

/* NOTES
* END NOTES */

static void                                /* R: no return value */
MMTR_MIMERegDestroy (
/* PARAMETERS */
    MMTR_MIME_REG_t ** mime_reg          /* I: MIME registration record to
destroy */
/* END PARAMETERS */
)
{
/* VARIABLES */
    CELLPTR curcell;                    /* current cell in handle list */
    MMTR_NOT_CB_ENTRY_t * cb_entry; /* notification callback entry */
/* END VARIABLES */

/* : sanity checks */
    assert (mime_reg != NULL);
    if (*mime_reg == NULL) {
        return;
    }

/* : destroy the mime type data */
    fs_give ((void **) &((*mime_reg)->mime_type));

/* : walk the registration list destroying callback entries */
    for (curcell = rc_ll_cell_first ((*mime_reg)->cb_list);
        curcell != NULL;
        curcell = rc_ll_cell_next (curcell)) {

/* : - get the mime registration from the registration list */
        cb_entry = (MMTR_NOT_CB_ENTRY_t *) rc_ll_cell_value (curcell);

/* : - destroy the mime registration record */
        fs_give ((void **) &cb_entry);
    }

/* : destroy the registration list */

```

```
rc_ll_list_destroy (&((*mime_reg)->cb_list), FALSE);

/* : destroy the mailbox entry */
fs_give ((void **) mime_reg);

/* : cleanup and return */
return;
}
/* END FUNCTION: MMTR_MIMERegDestroy */

/*
```

```

*/

/* FUNCTION: MMTR_MIMERegGet */

/* SYNOPSIS
 * Lookup and return the MIME registration that matches a MIME type in a mailbox
 * notification entry.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

static MMTR_MIME_REG_t *          /* R: MIME registration matching MIME
type */
MMTR_MIMERegGet (
/* PARAMETERS */
    MMTR_MLIST_ENTRY_t * entry,    /* I: mailbox notification entry to lookup
MIME type in */
    char * mime_type               /* I: MIME type to lookup registration for */
/* END PARAMETERS */
)
{
/* VARIABLES */
    CELLPTR curcell;               /* current cell in registration list */
    MMTR_MIME_REG_t * mime_reg;    /* MIME registration record */
/* END VARIABLES */

/* : sanity checks */
    assert (entry != NULL);
    assert (mime_type != NULL);

/* : initialization */

/* : walk the mailbox entry's notifier list trying to match */
    for (curcell = rc_ll_cell_first (entry->mime_notifiers);
         curcell != NULL;
         curcell = rc_ll_cell_next (curcell)) {

/* : - get the mime registration from the current cell */
        mime_reg = (MMTR_MIME_REG_t *) rc_ll_cell_value (curcell);

/* : - try to match the MIME type */
        if (strcmp (mime_reg->mime_type, mime_type) == 0) {
            return (mime_reg);
        }
    }
}

```

```
/* : CLAIM - if we get here then we did not find the MIME type that we were  
: looking for. */
```

```
/* : cleanup and return NULL */  
return (NULL);
```

```
}
```

```
/* END FUNCTION: MMTR_MIMERegGet */
```

```
/* END MODULE: message_monitor_API */
```

Module Name: Builder/mtrk/db/mtrkdb.c

/* MODULE: message_tracking_database_api */

/* COPYRIGHT

*

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

* All rights reserved.

*

* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,

* or a written license from an organization licensed by MessagingDirect Limited

* to grant such a license.

* END COPYRIGHT */

/* OVERVIEW

* This module contains the main function points for the Message Tracking database.

* END OVERVIEW */

/* PUBLIC DEPENDENCIES */

#include <stdio.h> /* standard C I/O definitions */

#include <stdlib.h> /* standard C library definitions */

#include <string.h> /* standard C string definitions */

#include <assert.h> /* standard C assertion definitions */

#include <sys/types.h> /* OS base type definitions */

#include <sys/stat.h> /* OS file statistic definitions */

#include "db.h" /* Berkeley DB database API */

#include "compat.h" /* Compatability library */

#include "eutility.h" /* Extended utility library */

#include "fs.h" /* memory manager definitions */

/* END PUBLIC DEPENDENCIES */

/* PRIVATE DEPENDENCIES */

#include "mtrk.h" /* Message Tracking Public API */

#include "mtrkdb.h" /* Message Tracking Database API */

#include "mtrkdb_utility.h" /* Message Tracking utility function prototypes */

/* : number of array entries in db_config array (last is null) */

#define DB_CONFIG_ENTRIES 4

/* : paths of various database files */

#define MTRKDB_PATH_DATA "data"

#define MTRKDB_PATH_LOGS "logs"

#define MTRKDB_PATH_TMP "tmp"

/* : DOC - directory structure is as follows:

```
: /.../database/
:      _db_ *           subsystem shared memory
:      data/
:      *.db            database tables
:      transaction_logs/
:      log.*          transaction logs
: */
```

/* END PRIVATE DEPENDENCIES */

/*

```

*/

/* FUNCTION: MTRKDB_DBCreate */

/* SYNOPSIS
* END SYNOPSIS */

/* NOTES
* END NOTES */

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if database instantiated
or else MTRK_FAIL */
MTRKDB_DBCreate (
/* PARAMETERS */
    char *db_path,           /* I: base path to databases */
    MTRKDB_CONTEXT_t ** db,  /* O: new message store database context */
    MTRKDB_VERBOSITY_t verbosity, /* I: verbosity level */
    MTRKDB_RECOVERY_t recovery /* I: recovery level */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRKDB_CONTEXT_t *local_db; /* working copy of db context */
    DB_ENV *env;                /* local db environment struct */
    DB_INFO info;               /* db specific options struct */
    char *config[DB_CONFIG_ENTRIES]; /* db configuration paths */
    char *log_dir;              /* db log directory */
    char *tmp_dir;              /* db tmp directory */
    char *data_dir;             /* db data directory */
    int i, r;                   /* counter and generic return value */
    u_int32_t flags;            /* flags for db_open() */
    int mode;                   /* mode of database files */
    char *c;                   /* temp string pointer */
    MTRKDB_RECOVERY_t recovery_level; /* local recovery level */
    u_int32_t recover_flag;     /* recovery flag for db_appinit */
    MTRK_RETURN_CODE_t return_value; /* function return value */
/* END VARIABLES */

/* : sanity checks */
    assert (db_path != NULL);
    assert (db != NULL);

/* : make sure that verbosity and recovery settings are valid */
    assert (verbosity == MTRKDB_VERBOSE_NONE ||
            verbosity == MTRKDB_VERBOSE_STDERR ||
            verbosity == MTRKDB_VERBOSE_SYSLOG);

```



```

assert (recovery == MTRKDB_RECOVER_DEFAULT ||
        recovery == MTRKDB_RECOVER_NONE ||
        recovery == MTRKDB_RECOVER_PROGRESSIVE ||
        recovery == MTRKDB_RECOVER_RECOVER ||
        recovery == MTRKDB_RECOVER_FATAL);

/* : initialization */
log_dir = NULL;
tmp_dir = NULL;
data_dir = NULL;
flags = 0;
recover_flag = 0;
local_db = NULL;
env = NULL;
mode = 0;
r = 0;
c = NULL;
return_value = MTRK_SUCC;          /* assume success */

/* : initialize local recovery level and set default policy */
recovery_level = recovery;
if (recovery_level == MTRKDB_RECOVER_DEFAULT) {
    recovery_level = MTRKDB_RECOVER_PROGRESSIVE;
}

/* : set initial flags and mode for database opening */
flags = DB_CREATE | DB_THREAD;
mode = S_IREAD | S_IWRITE;

/* : make sure path exists for database files */
if (mtrkdb_dir_validate(db_path) == FAIL) {
    fprintf(stderr,
            "Cannot create database directory or directory not writable:
%s\n",db_path);
    cleanup_return(MTRK_FAIL);
}

/* : construct and validate transaction log path */
log_dir = EU_PathConstruct(db_path,MTRKDB_PATH_LOGS);
if (mtrkdb_dir_validate(log_dir) == MTRK_FAIL) {
    fprintf(stderr,
            "Cannot create trx log directory or directory not writable: %s\n",log_dir);
    cleanup_return(MTRK_FAIL);
}
#endif DEADCODE
if (log_dir == NULL) {

```

```

        if (verbosity == MTRKDB_VERBOSE_SYSLOG) {
            mtrkdb_error ("MTRKDB", "Cannot determine transaction-log-directory");
        }
        else if (verbosity == MTRKDB_VERBOSE_STDERR) {
            fprintf (stderr, "Cannot determine transaction-log-directory\n");
        }
        cleanup_return (MTRK_FAIL);
    }
#endif

/* : construct and validate tmp dir */
tmp_dir = EU_PathConstruct(db_path,MTRKDB_PATH_TMP);
if (mtrkdb_dir_validate(tmp_dir) == MTRK_FAIL) {
    fprintf (stderr,
            "Cannot create tmp directory or directory not writable: %s\n",tmp_dir);
    cleanup_return(MTRK_FAIL);
}

/* : construct and validate database path */
data_dir = EU_PathConstruct(db_path,MTRKDB_PATH_DATA);
if (mtrkdb_dir_validate(data_dir) == MTRK_FAIL) {
    fprintf (stderr,
            "Cannot create data directory or directory not writable: %s\n",data_dir);
    cleanup_return(MTRK_FAIL);
}

/* : create list of db file paths */
config[0] = (char *) fs_get (strlen ("DB_TMP_DIR ") + strlen
(MTRKDB_PATH_TMP) + 1);
sprintf (config[0], "DB_TMP_DIR %s", MTRKDB_PATH_TMP);
config[1] = (char *) fs_get (strlen ("DB_LOG_DIR ") + strlen
(MTRKDB_PATH_LOGS) + 1);
sprintf (config[1], "DB_LOG_DIR %s", MTRKDB_PATH_LOGS);
config[2] = (char *) fs_get (strlen ("DB_DATA_DIR ") + strlen
(MTRKDB_PATH_DATA) + 1);
sprintf (config[2], "DB_DATA_DIR %s", MTRKDB_PATH_DATA);
config[DB_CONFIG_ENTRIES - 1] = NULL;

/* : remove trailing '/' from all paths */
for (i = 0; i < (DB_CONFIG_ENTRIES - 1); i++) {
    c = config[i] + strlen (config[i]) - 1;
    while (*c == '/') {
        *c = '\0';
        c--;
    }
    /*mtrkdb_error ("MTRKDB-DEBUG", config[i]);*/
}

```

```

/* : allocate local context*/
local_db = (MTRKDB_CONTEXT_t *) fs_get (sizeof (MTRKDB_CONTEXT_t));

/* : allocate and init DB_ENV structure */
env = (DB_ENV *) fs_get (sizeof (DB_ENV));
memset (env, 0, sizeof (DB_ENV));
env->db_errpfx = "MTRKDB";
env->mp_size = 1048576;          /* use 1Mb cache */
switch (verbosity) {
case MTRKDB_VERBOSE_STDERR:
    env->db_verbose = 1;          /* verbosity on */
    env->db_errfile = stderr;     /* output internal errs to stderr */
    break;
case MTRKDB_VERBOSE_SYSLOG:
    env->db_verbose = 1;          /* verbosity on */
    env->db_errcall = &mtrkdb_error; /* output internal errs to syslog */
    break;
case MTRKDB_VERBOSE_NONE:
    env->db_verbose = 0;          /* verbosity off */
    break;
}

/* : set the flags for db_appinit according to recovery level */
/* : DOC - do nothing for MTRKDB_RECOVER_NONE and
MTRKDB_RECOVER_PROGRESSIVE */
if (recovery_level == MTRKDB_RECOVER_RECOVER) {
    recover_flag = DB_RECOVER;
}
else if (recovery_level == MTRKDB_RECOVER_FATAL) {
    recover_flag = DB_RECOVER_FATAL;
}

/* : initialize the main database system */
r = db_appinit (db_path, config, env,
                flags | DB_INIT_LOCK | DB_INIT_LOG |
                DB_INIT_MPOOL | DB_INIT_TXN | recover_flag);

/* : in case of fatal error, initialize with recovery */
if (recovery_level == MTRKDB_RECOVER_PROGRESSIVE && r ==
DB_RUNRECOVERY) {
    if (verbosity == MTRKDB_VERBOSE_SYSLOG) {
        mtrkdb_error ("MTRKDB", "Appinit error found, attempting to recover ...");
    }
    else if (verbosity == MTRKDB_VERBOSE_STDERR) {
        fprintf (stderr, "Appinit error found, attempting to recover ...\n");
    }
}

```

```

    }
    r = db_appinit (db_path, config, env,
                    flags | DB_INIT_LOCK | DB_INIT_LOG |
                    DB_INIT_MPOOL | DB_INIT_TXN | DB_RECOVER);
}

/* : free up the config array */
for (i = 0; i < DB_CONFIG_ENTRIES; i++) {
    fs_give ((void **) &config[i]);
}

/* : fail if database refused to initialize */
if (r != 0) {
    cleanup_return(MTRK_FAIL);
}

/* : init DB_INFO structure */
memset (&info, 0, sizeof (info));
info.db_cachesize = 0;          /* 0 means use default size */
info.db_lorder = 0;            /* 0 means use host byte order */
info.db_pagesize = 0;          /* thought of using 512, but 0 defaults to fs
blocksize */
info.db_malloc = mtrkdb_fs_get; /* use fs_get() for memory allocation */
info.bt_compare = NULL;        /* NULL means compare keys lexically */
info.bt_minkey = 0;            /* 0 defaults to minimum of 2 keys per page */
info.bt_prefix = NULL;        /* NULL means compare keys lexically */

/* : open each of the databases and add to handle array */

/* :- open the main data table which store the status records */
r = db_open (MTRKDB_DB_NAME_DATA,
             DB_RECNO, flags, mode, env, &info,
             &(local_db->db_handles[MTRKDB_DB_TYPE_DATA]));
if (r != 0) {
    cleanup_return(MTRK_FAIL);
}

/* :- DOC - for the index tables, support retrieval by record num and do NOT
:- allow duplicate records */
info.flags = DB_RECNUM;

/* :- open msgid index table */
r = db_open (MTRKDB_DB_NAME_MSGID,
             DB_BTREE, flags, mode, env, &info,
             &(local_db->db_handles[MTRKDB_DB_TYPE_MSGID]));
if (r != 0) {

```

```

        cleanup_return(MTRK_FAIL);
    }

/* :- open status_code index table */
r = db_open (MTRKDB_DB_NAME_STATUS,
             DB_BTREE, flags, mode, env, &info,
             &(local_db->db_handles[MTRKDB_DB_TYPE_STATUS]));
if (r != 0) {
    cleanup_return(MTRK_FAIL);
}

/* :- open user index table */
r = db_open (MTRKDB_DB_NAME_USER,
             DB_BTREE, flags, mode, env, &info,
             &(local_db->db_handles[MTRKDB_DB_TYPE_USER]));
if (r != 0) {
    cleanup_return(MTRK_FAIL);
}

local_db->last_error = MTRKDB_ERROR_NONE;
local_db->db_env = env;
*db = local_db;

CLEANUP:
/* : clean tmp and log dir vars */
fs_give ((void **) &log_dir);
fs_give ((void **) &tmp_dir);
fs_give ((void **) &data_dir);

if (return_value == MTRK_FAIL) {
    MTRKDB_DBShutdown (&local_db);
}
return (return_value);
}

/* END FUNCTION: MTRKDB_DBCreate */

/*

```

```

*/

/* FUNCTION: MTRKDB_DBShutdown */

/* SYNOPSIS
 * Shutdown (close) database tables and deallocate all database context resources.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

void                                     /* R: no return value */
MTRKDB_DBShutdown (
/* PARAMETERS */
    MTRKDB_CONTEXT_t ** db             /* U: message tracking database context to
destroy */
/* END PARAMETERS */
)
{
/* VARIABLES */
    int i;                             /* counting var */
/* END VARIABLES */

/* : sanity checks */
    assert (db != NULL);
    if (*db == NULL) {
        return;
    }

/* : close each database and deallocate it from context */
    for (i = 0; i < MTRKDB_NUM_DB_TYPES; i++) {
        if ((*db)->db_handles[i] != NULL) {
            (*db)->db_handles[i]->close ((*db)->db_handles[i], 0);
        }
    }

/* : close database subsystems */
    if ((*db)->db_env != NULL) {
        db_appexit ((*db)->db_env);
    }

/* : free up rest on context and context itself */
    if ((*db)->db_env != NULL) {
        fs_give ((void **) &((*db)->db_env));
    }
}

```

```
    if (*db != NULL) {  
        fs_give ((void **) db);  
    }  
  
    /* : cleanup and return */  
    return;  
}  
  
/* END FUNCTION: MTRKDB_DBShutdown */  
  
/*
```

```

*/

/* FUNCTION: MTRKDB_RegisterThread */

/* SYNOPSIS
 * Register current thread as database consumer.
 * END SYNOPSIS */

void                                     /* R: no return value */
MTRKDB_RegisterThread (
/* PARAMETERS */
    MTRKDB_CONTEXT_t * db               /* I: message tracking database
context */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
    assert (db != NULL);

/* : initialization */

/* : cleanup and return */
    return;
}

/* END FUNCTION: MTRKDB_RegisterThread */

/*

```



```

*/

/* FUNCTION: MTRKDB_UnregisterThread */

/* SYNOPSIS
 * Unregister current thread as database consumer.
 * END SYNOPSIS */

void                                     /* R: no return value */
MTRKDB_UnregisterThread (
/* PARAMETERS */
    MTRKDB_CONTEXT_t * db               /* I: message tracking database
context */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
    assert (db != NULL);

/* : initialization */

/* : cleanup and return */
    return;
}

/* END FUNCTION: MTRKDB_UnregisterThread */

/*

```

```

*/

/* FUNCTION: MTRKDB_GetLastError */

/* SYNOPSIS
* Get the last error encountered by the thread in the database.
* END SYNOPSIS */

/* NOTES
* END NOTES */

MTRKDB_ERROR_CODE_t                               /* R: last error code encountered by
thread in database */
MTRKDB_GetLastError (
/* PARAMETERS */
    MTRKDB_CONTEXT_t * db                         /* I: message tracking database
context */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
    assert (db != NULL);

/* : cleanup and return the error code */
    return (db->last_error);
}
/* END FUNCTION: MTRKDB_GetLastError */

/* END MODULE: message_tracking_database_api */

```

Module Name: Builder/mtrk/db/mtrkdb.h

/* MODULE: message_tracking_database_api */

/* COPYRIGHT

*

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

* All rights reserved.

*

* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,

* or a written license from an organization licensed by MessagingDirect Limited

* to grant such a license.

* END COPYRIGHT */

/* OVERVIEW

* This module contains the public API for the Message Tracking database.

* END OVERVIEW */

/* multiple inclusion protection */

#ifndef _message_tracking_database_api

#define _message_tracking_database_api

/* PUBLIC DEPENDENCIES */

#include "db.h"

/* sleepycat DB API */

#include "mtrkdb_table.h"

/* END PUBLIC DEPENDENCIES */

/* : DOC - definitions for MTRKDB error codes */

typedef enum {

 MTRKDB_ERROR_NONE = 0, /* no error */

 MTRKDB_ERROR_INVVAL = 1, /* invalid input value */

 MTRKDB_ERROR_KEYNOTFOUND, /* key not found in db */

 MTRKDB_ERROR_NULLVALUE, /* no corresponding value found for

key */

 MTRKDB_ERROR_DBERR, /* db operational error */

 MTRKDB_ERROR_MALFREC /* malformed record (PIF error) */

} MTRKDB_ERROR_CODE_t;

/* : DOC - MTRKDB_Create() verbosity levels */

typedef enum {

 MTRKDB_VERBOSE_NONE = 0, /* no verbosity */

 MTRKDB_VERBOSE_STDERR, /* all db internal errors to stderr */

 MTRKDB_VERBOSE_SYSLOG /* all db internal errors to syslog */

} MTRKDB_VERBOSITY_t;

```

/* : DOC - MTRKDB_Create() recovery levels */
typedef enum {
    MTRKDB_RECOVER_DEFAULT,          /* use default level as defined in
MTRKDB_Create() */
    MTRKDB_RECOVER_NONE,             /* do not initialize the DB with recovery */
    MTRKDB_RECOVER_PROGRESSIVE,      /* if init without fails then reinit with
recovery */
    MTRKDB_RECOVER_RECOVER,          /* init DB with recovery */
    MTRKDB_RECOVER_FATAL             /* init DB with catastrophic recovery */
} MTRKDB_RECOVERY_t;

```

```

/* : DOC - open database context. This structure maintains open contexts for all
: the database files that we use in the message tracking subsystem. We want to open
and close
: the contexts for the various databases exactly once per process because opening
: and closing is an expensive operation. */

```

```

typedef struct {
    DB_ENV *db_env;                  /* database environment */
    DB *db_handles[MTRKDB_NUM_DB_TYPES]; /* open database handles */
    MTRKDB_ERROR_CODE_t last_error; /* last error code */
} MTRKDB_CONTEXT_t;

```

```

/* message status codes */

```

```

typedef enum {
    MTRK_SC_INVALID = 0,             /* so that zero is not a valid status code */
    MTRK_SC_SUBMITTED,               /* message has been injected into
mail transfer system */
    MTRK_SC_RELAYED,                 /* message transferred from one host to
another */
    MTRK_SC_DELIVERED,               /* message has reached the destination
mailbox */
    MTRK_SC_FORWARDED,               /* message has been forwarded to a
mail client (ie: mail list exploder */
    MTRK_SC_BLACKHOLE,               /* message has been relayed to a
non-DSN capable host */
    MTRK_SC_DELAYED,                 /* message has been temporarily delayed at
the specified host */
    MTRK_SC_REJECTED,                /* message not able to be delivered to
destination mailbox */
    MTRK_SC_SEEN                     /* message has been opened by user using an MUA
*/
} MTRK_STATUS_CODE_t;

```

```

#define MTRK_STATUS_CODE_FIRST MTRK_SC_SUBMITTED /* for validity
checking of status codes */
#define MTRK_STATUS_CODE_LAST MTRK_SC_SEEN

```

```

/* to convert message status codes to strings */
#define MTRK_SC_SUBMITTED_STRING "Submitted"
#define MTRK_SC_RELAYED_STRING "Relayed"
#define MTRK_SC_DELIVERED_STRING "Delivered"
#define MTRK_SC_FORWARDED_STRING "Forwarded"
#define MTRK_SC_BLACKHOLE_STRING "Non DSN Aware"
#define MTRK_SC_DELAYED_STRING "Delayed"
#define MTRK_SC_REJECTED_STRING "Rejected"
#define MTRK_SC_SEEN_STRING "Seen"
#define MTRK_SC_INVALID_STRING "Not recognized"

/* message status record layout */
typedef struct {
    char * msgid; /* unique message identifier */
    MTRK_STATUS_CODE_t status; /* status of message */
    char * host; /* host to which status record applies */
    char * to_host; /* used only by RELAYED state records */
    char * user; /* used only by SEEN state records */
    char * description; /* status reason description (optional) */
    time_t timestamp; /* creation timestamp for status record */
} MTRK_STATUS_REC_t;

/* : DOC - callback function to handle a match from the database.
   : The calling application must provide a function of this
   : type for the callback function. */

typedef int /* R: zero if successful, non-zero if not */
(MTRKDB_REC_MATCH_CB_ft) ( /* D: callback function to handle a
match from database */
    void * key, /* I: full key which matched submitted criteria */
    MTRKDB_DB_TYPE_t db_type, /* I: which index table the key is from */
    MTRK_STATUS_REC_t * rec, /* I: record which matched submitted
criteria */
    void * data /* I: callback context data */
);

typedef int /* R: zero if successful, non-zero if not */
(MTRKDB_KEY_MATCH_CB_ft) ( /* D: callback function to handle a
key match */
    void * key, /* I: full key which matched submitted criteria */
    int key_length, /* I: length of key returned, in bytes */
    MTRKDB_DB_TYPE_t db_type, /* I: which index table the key is from */
    void * data /* I: callback context data */
);

```

```

/* function prototypes */

/* database-specific functions */
MTRK_RETURN_CODE_t MTRKDB_DBCreate(
    char *db_path,                /* I: base path to databases */
    MTRKDB_CONTEXT_t ** db,       /* O: message tracking database context */
    MTRKDB_VERBOSITY_t verbosity, /* I: verbosity level */
    MTRKDB_RECOVERY_t recovery    /* I: recovery level */
);

MTRK_RETURN_CODE_t MTRKDB_DBInit(
    MTRKDB_CONTEXT_t * db         /* U: message tracking database
context */
);

void MTRKDB_DBShutdown(
    MTRKDB_CONTEXT_t ** db        /* U: message tracking database context */
);

void                                /* R: no return value */
MTRKDB_RegisterThread (
    MTRKDB_CONTEXT_t * db         /* I: message tracking database
context */
);

void                                /* R: no return value */
MTRKDB_UnregisterThread (
    MTRKDB_CONTEXT_t * db         /* I: message tracking database
context */
);

MTRKDB_ERROR_CODE_t                /* R: last error code encountered by
thread in database */
MTRKDB_GetLastError (
    MTRKDB_CONTEXT_t * db         /* I: message tracking database
context */
);

/* record-specific functions */
MTRK_RETURN_CODE_t                /* R: MTRK_SUCC if user is added else
MTRK_FAIL */
MTRKDB_RecAdd (
    MTRKDB_CONTEXT_t * db,        /* D: add a record to the database */
    MTRK_STATUS_REC_t *           /* I: message tracking database context */
    /* I: status record to be added */
);

```

```

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if records fetched,
MTRK_FAIL otherwise */
MTRKDB_RecFetchByMsgid (     /* D: fetch records with given msgid */
    MTRKDB_CONTEXT_t * db,   /* I: message tracking database context */
    char *msgid_key,          /* I: msgid key of records to fetch */
    int key_length,           /* I: length of msgid_key in bytes */
    MTRKDB_REC_MATCH_CB_ft match_cb, /* I: callback function to call when a
match found */
    void * data               /* I: callback context data passed to callback
function */
);

```

```

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if records fetched,
MTRK_FAIL otherwise */
MTRKDB_RecFetchByStatus(     /* D: fetch records with given status code */
    MTRKDB_CONTEXT_t * db,   /* I: message tracking database context */
    void * status_key,        /* I: status key of records to fetch */
    int key_length,           /* I: length of status_key in bytes */
    MTRKDB_REC_MATCH_CB_ft match_cb, /* I: callback function to call when a
match found */
    void * data               /* I: callback context data passed to callback
function */
);

```

```

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if records fetched,
MTRK_FAIL otherwise */
MTRKDB_RecFetchByUser (     /* D: fetch records with given user */
    MTRKDB_CONTEXT_t * db,   /* I: message tracking database context */
    char *user_key,          /* I: user key of records to fetch */
    int key_length,           /* I: length of user_key in bytes */
    MTRKDB_REC_MATCH_CB_ft match_cb, /* I: callback function to call when a
match found */
    void * data               /* I: callback context data passed to callback
function */
);

```

/* Key fetch functions */

```

MTRK_RETURN_CODE_t
MTRKDB_KeyFetchByMsgid (
    MTRKDB_CONTEXT_t * db,   /* I: message tracking database context */
    char *msgid_key,          /* I: msgid of keys to fetch */
    int key_length,           /* I: length of msgid_key in bytes */
    MTRKDB_KEY_MATCH_CB_ft match_cb, /* I: callback function to call when a
match found */

```

```

    void * data                /* I: callback context data passed to callback
function */
);

MTRK_RETURN_CODE_t
MTRKDB_KeyFetchByUser (
    MTRKDB_CONTEXT_t * db,      /* I: message tracking database context */
    char *user_key,             /* I: user key of keys to fetch */
    int key_length,             /* I: length of user_key in bytes */
    MTRKDB_KEY_MATCH_CB_ft match_cb, /* I: callback function to call when a
match found */
    void * data                /* I: callback context data passed to callback
function */
);

MTRK_RETURN_CODE_t
MTRKDB_KeyFetchByStatus (
    MTRKDB_CONTEXT_t * db,      /* I: message tracking database context */
    void * status_key,          /* I: status key of records to fetch */
    int key_length,             /* I: length of status_key in bytes */
    MTRKDB_KEY_MATCH_CB_ft match_cb, /* I: callback function to call when a
match found */
    void * data                /* I: callback context data passed to callback
function */
);

/* utility functions */

MTRK_STATUS_REC_t * MTRK_StatusRecNew (void);
void MTRK_StatusRecDestroy (MTRK_STATUS_REC_t **rec);

#endif /* multiple inclusion protection */

/* END MODULE: message_tracking_database_api */

```


Module Name: Builder/mtrk/report/mtrkrep.c

/* MODULE: message_tracking_reporting_api */

/* COPYRIGHT

*

* Copyright (c) MessagingDirect Limited, Edmonton, Canada

* All rights reserved.

*

* Acquisition and use of this software and related materials for any

* purpose requires a written license agreement from MessagingDirect Limited,

* or a written license from an organization licensed by MessagingDirect Limited

* to grant such a license.

* END COPYRIGHT */

/* OVERVIEW

* This module contains the main API points for the Message Tracking Reporting subsystem.

* END OVERVIEW */

/* PUBLIC DEPENDENCIES */

#include <stdio.h> /* standard C I/O definitions */

#include <stdlib.h> /* standard C library definitions */

#include <string.h> /* standard C string definitions */

#include <assert.h> /* standard C assertion definitions */

#include <time.h> /* standard C time definitions */

#include <stdarg.h> /* standard C variable argument definitions */

#include <ctype.h> /* standard C type definitions */

#include "db.h"

#include "compat.h" /* Compatability library */

#include "fs.h" /* free storage memory API */

#include "eutility.h" /* Extended utility library */

#include "mtrk.h" /* Message Tracking Public API */

#include "mtrkdb.h" /* Message Tracking Database public API */

#include "mtrkrep.h" /* Message Tracking Reporting API */

/* END PUBLIC DEPENDENCIES */

/* PRIVATE DEPENDENCIES */

```
static int mtrkrep_rec_match_callback (void * full_key,  
                                       MTRKDB_DB_TYPE_t db_type,  
                                       MTRK_STATUS_REC_t * rec,  
                                       void * data);
```

```

static int mtrkrep_key_match_callback (void * full_key,
                                       int key_length,
                                       MTRKDB_DB_TYPE_t db_type,
                                       void * data);

static void  mtrkrep_status_string_translate_to_code (char *status_string,
MTRK_STATUS_CODE_t *status);
static void  mtrkrep_status_code_translate_to_string (MTRK_STATUS_CODE_t status,
char **status_string);
static BOOLEAN mtrkrep_partial_key_match (void *full_key,
                                       void *partial_key,
                                       int partial_key_length,
                                       MTRKDB_DB_TYPE_t db_type);

void
MTRKREP_FullKeySplit (
    MTRKDB_DB_TYPE_t db_type,          /* I: which index table is this key from */
    void * full_key,                   /* I: full key from database */
    int  full_key_length,               /* I: length of full_key in bytes */
    char **r_key_part1,                /* OA: returned string containing part1 of the record
key */
    char **r_key_part2                 /* OA: returned string containing part2 of the record
key, an alphanumeric recno */
);

typedef struct {
    char *original_key;                /* original search key */
    int key_length;                    /* length in bytes of original_key */
    int pagesize;                      /* how many recs per page */
    int rec_count;                     /* how many recs we've fetched so far */
    MTRKREP_PHP_REC_MATCH_CB_ft php_cb; /* PHP callback function to
pass formatted record to */
    void * php_cb_data;                /* PHP data to pass to PHP callback */
} MTRKREP_REC_MATCH_CB_CONTEXT_t;

typedef struct {
    char *original_key;                /* original search key */
    int key_length;                    /* length in bytes of original_key */
    int pagesize;                      /* how many keys per page */
    int key_count;                     /* how many keys we've fetched so far */
    MTRKREP_PHP_KEY_MATCH_CB_ft php_cb; /* PHP callback function to
pass key to */
    void * php_cb_data;                /* PHP data to pass to PHP callback */
} MTRKREP_KEY_MATCH_CB_CONTEXT_t;

#define MTRKREP_ALPHA_RECNO_LEN 21

```

```
FILE *g_debug_file = NULL;  
BOOLEAN g_debug_logging = FALSE;
```

```
/* END PRIVATE DEPENDENCIES */
```

```
/*
```

```

*/

/* FUNCTION: MTRKREP_Init */

/* SYNOPSIS
 * Initialize the Message Tracking Reporting API by opening up a context to the
 * database.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if database opened
successfully */
MTRKREP_Init (
/* PARAMETERS */
    char *db_path,           /* I: path to database files, cannot be NULL */
#ifdef USE_MTRKREP
    MTRKREP_DATABASE_CONTEXT_t ** db, /* OA: message tracking database
context */
#else
    MTRKDB_CONTEXT_t ** db,         /* OA: message tracking database context */
#endif
    BOOLEAN debug,              /* I: 1 = turn on debug logging, 0 = no debug
logging */
    char *debug_file_path      /* I: path to the debug log file if debug = 1 */
/* END PARAMETERS */
)
{
/* VARIABLES */
    time_t now;
    char * timestamp = NULL;
/* END VARIABLES */

/* : open the debug log file */
    if (debug == 1) {
        if (debug_file_path != NULL) {
            g_debug_file = fopen(debug_file_path,"a");
            if (g_debug_file == NULL) {
                fprintf(stderr,"MTRKREP_Init: Unable to open debug_file
%s\n",debug_file_path);
            } else {
                g_debug_logging = TRUE;
                now = time(0);
                timestamp = ctime (&now);
            }
        }
    }
}

```

```

/* convert trailing \n of time stamp string to \0 */
    timestamp[strlen(timestamp) - 1] = '\0';
    fprintf(stderr,"MTRKREP_Init: debug logging started in file %s at
%s\n",debug_file_path,timestamp);
    mtrkrep_debug("MTRKREP_Init: Debug logging started at
%s\n",timestamp);
    }
}
else {
    mtrkrep_debug("MTRKREP_Init: Unable to open debug_file, no path given.
Debug logging disabled.\n");
}
}

/* : sanity checks */
assert(db_path != NULL);
assert(db != NULL);

/* : initialization */
/* : create/initialize the database */
if (MTRKDB_DBCreate (db_path,
                    (MTRKDB_CONTEXT_t **)db,
                    MTRKDB_VERBOSE_STDERR,
                    MTRKDB_RECOVER_DEFAULT) == MTRK_FAIL) {
    mtrkrep_debug("MTRKREP_Init: Error initializing database.\n");
    return(MTRK_FAIL);
}

/* : cleanup and return */
return (MTRK_SUCC);
}

/* END FUNCTION: MTRKREP_Init */

/*

```

```

*/

/* FUNCTION: MTRKREP_Shutdown */

/* SYNOPSIS
 * Initialize the Message Tracking Reporting API by opening up a context to the
 * database.
 * END SYNOPSIS */

/* NOTES
 * END NOTES */

void                                     /* R: no return type */
MTRKREP_Shutdown (
/* PARAMETERS */
#ifdef USE_MTRKREP
    MTRKREP_DATABASE_CONTEXT_t ** db    /* OA: message tracking database
context */
#else
    MTRKDB_CONTEXT_t ** db              /* OA: message tracking database context */
#endif
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* debugging */
    mtrkrep_debug("MTRKREP_Shutdown: function entry\n");

/* : sanity checks */

/* : initialization */
/* : create/initialize the database */
    MTRKDB_DBShutdown ((MTRKDB_CONTEXT_t **)db);

    if (g_debug_file != NULL)
        fclose(g_debug_file);
/* : cleanup and return */
    return;
}

/* END FUNCTION: MTRKREP_Shutdown */

/*

```

```

*/

/* FUNCTION: MTRKREP_RecList */

/* SYNOPSIS
* Fetch a list of records with the given full key value using index table defined by
key_type.
* Call the php_rec_match_cb when a match is found.
* END SYNOPSIS */

/* NOTES
* full_key_value is expected to be a full key. It should contain a key value as part 1 and
a binary recno as part 2.
* The full_key_value buffer should be KEY_LEN+MTRKDB_RECNO_LEN bytes
long.
* END NOTES */

MTRK_RETURN_CODE_t          /* R: MTRK_SUCC if records fetched
successfully */
MTRKREP_RecList (
/* PARAMETERS */
#ifdef USE_MTRKREP
    MTRKREP_DATABASE_CONTEXT_t * db,    /* I: message tracking database
context */
#else
    MTRKDB_CONTEXT_t * db,              /* I: message tracking database context */
#endif
    char *key_type,                    /* I: type of key attribute to search on */
    void *full_key_value,               /* I: value of key attribute to match */
    int num_recs_to_fetch,              /* I: number of records to fetch */
    MTRKREP_PHP_REC_MATCH_CB_ft php_rec_match_cb, /* I: callback
function */
    void *php_rec_match_cb_context     /* I: callback context data passed to
callback */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRK_RETURN_CODE_t return_code; /* function return code */
    MTRKREP_REC_MATCH_CB_CONTEXT_t *data;
/* END VARIABLES */

/* : debug */
    mtrkrep_debug("MTRKREP_RecList: function entry\n");

/* : sanity checks */

```

```

    assert(db != NULL);

/* : initialization */
    data = NULL;

    data = (MTRKREP_REC_MATCH_CB_CONTEXT_t
*)fs_get(sizeof(MTRKREP_REC_MATCH_CB_CONTEXT_t));

    data->pagesize = num_recs_to_fetch;
    data->rec_count = 0;
    data->php_cb = php_rec_match_cb;
    data->php_cb_data = php_rec_match_cb_context;

    switch (key_type[0]) {
        case 'M': /* "MSGID" */
        case 'm':
            mtrkrep_debug("MTRKREP_RecList: calling RecFetchByMsgid with key
%s\n",full_key_value);
/* : pull out the string msgid part from the full key */
            data->key_length = MTRKDB_MSGID_LEN;
            data->original_key = fs_get(data->key_length + 1);
            memset(data->original_key,'\0',data->key_length + 1);
            memcpy(data->original_key,full_key_value,data->key_length);

/* : fetch the records based on the full key (msgid + recno) */
            return_code = MTRKDB_RecFetchByMsgid ((MTRKDB_CONTEXT_t *)db,
                full_key_value,

MTRKDB_MSGID_LEN+1+MTRKDB_RECNO_LEN,
                mtrkrep_rec_match_callback,
                data);

            break;
        case 'U': /* "USER" */
        case 'u':
            mtrkrep_debug("MTRKREP_RecList: calling RecFetchByUser with key
%s\n",full_key_value);

/* : pull out the string user part from the full key */
            data->key_length = MTRKDB_USER_LEN;
            data->original_key = fs_get(data->key_length + 1);
            memset(data->original_key,'\0',data->key_length + 1);
            memcpy(data->original_key,full_key_value,data->key_length);

/* : fetch the records based on the full key (user + recno) */
            return_code = MTRKDB_RecFetchByUser ((MTRKDB_CONTEXT_t *)db,
                full_key_value,

```



```

MTRKDB_USER_LEN+MTRKDB_RECNO_LEN,
                                mtrkrep_rec_match_callback,
                                data);

    break;
case 'S': /* "STATUS" */
case 's': {
    MTRK_STATUS_CODE_t status;
    MTRKDB_RECNO_t recno;
/* : pull out the string msgid part from the full key */
    memcpy(&status,full_key_value,MTRKDB_STATUS_LEN);
    memcpy(&recno,full_key_value+MTRKDB_STATUS_LEN,MTRKDB_RECNO_LEN);
    mtrkrep_debug("MTRKREP_RecList: calling RecFetchByStatus with status %d,
recno %d\n",status,recno);
    data->key_length = MTRKDB_STATUS_LEN;
    data->original_key = fs_get(data->key_length);
    memset(data->original_key,'\0',data->key_length + 1);
    memcpy(data->original_key,full_key_value,data->key_length);

/* : fetch the records based on the full key (status + recno) */
    return_code = MTRKDB_RecFetchByStatus ((MTRKDB_CONTEXT_t *)db,
                                full_key_value,

MTRKDB_STATUS_LEN+MTRKDB_RECNO_LEN,
                                mtrkrep_rec_match_callback,
                                data);

    break;
}
default: /* unknown, cannot continue */
    return_code = MTRK_FAIL;
    break;
}

/* : cleanup and return */
fs_give((void **)&data->original_key);
fs_give((void **)&data);
mtrkrep_debug("MTRKREP_RecList: function exit\n");
return(return_code);
}

/* END FUNCTION: MTRKREP_RecList */

/*

```

*/

/* FUNCTION: MTRKREP_KeyList */

/* SYNOPSIS

* END SYNOPSIS */

/* NOTES

* This routine always assumes a PARTIAL key match. That is, the key_value provided is only

* part one of a 2-part key. The binary recno part is not included.

* The length of the key to match is determined by strlen(key_value) in the case of MSGID

* and USER. In the case of status, it is always MTRKDB_STATUS_LEN (sizeof(MTRK_STATUS_CODE_t)).

* In contrast, a FULL key match would provide a key which includes the recno of the

* record to be fetched. A full key match is used in the RecList function above.

* END NOTES */

MTRK_RETURN_CODE_t /* R: MTRK_SUCC if records fetched
successfully */

MTRKREP_KeyList (

/* PARAMETERS */

#if USE_MTRKREP

 MTRKREP_DATABASE_CONTEXT_t * db, /* I: message tracking database
context */

#else

 MTRKDB_CONTEXT_t * db, /* I: message tracking database context */

#endif

 char *key_type, /* I: type of key attribute to search on */

 void *key_part1, /* I: value of key attribute to match */

 int pagesize, /* I: return keys 1, pagesize+1, ((pagesize*2) + 1)

etc */

 MTRKREP_PHP_KEY_MATCH_CB_ft php_key_match_cb, /* I: callback
function */

 void * php_key_match_cb_context /* I: callback context data passed to
callback */

/* END PARAMETERS */

)

{

/* VARIABLES */

 MTRK_RETURN_CODE_t return_code; /* function return code */

 MTRKREP_KEY_MATCH_CB_CONTEXT_t *data;

 MTRK_STATUS_CODE_t status;

/* END VARIABLES */

```

/* : debug */
mtrkrep_debug("MTRKREP_KeyList: function entry\n");

/* : sanity checks */
assert(db != NULL);

mtrkrep_debug("MTRKREP_KeyList: key_type %s, key_part1 %s, pagesize
%d\n",key_type,key_part1,pagesize);

/* : initialization */
data = NULL;

data = (MTRKREP_KEY_MATCH_CB_CONTEXT_t
*)fs_get(sizeof(MTRKREP_KEY_MATCH_CB_CONTEXT_t));

data->key_count = 0;
data->pagesize = pagesize;
data->php_cb = php_key_match_cb;
data->php_cb_data = php_key_match_cb_context;

/* : call appropriate routine based on key type */
switch (key_type[0]) {
case 'M': /* "MSGID" */
case 'm':
mtrkrep_debug("MTRKREP_KeyList: calling KeyFetchByMsgid with key
%s\n",key_part1);
data->original_key = EU_StrDup(key_part1);
data->key_length = strlen(key_part1);
return_code = MTRKDB_KeyFetchByMsgid ((MTRKDB_CONTEXT_t *)db,
key_part1,
data->key_length,
mtrkrep_key_match_callback,
data);

break;
case 'U': /* "USER" */
case 'u':
mtrkrep_debug("MTRKREP_KeyList: calling KeyFetchByUser with key
%s\n",key_part1);
data->original_key = EU_StrDup(key_part1);
data->key_length = strlen(key_part1);
return_code = MTRKDB_KeyFetchByUser ((MTRKDB_CONTEXT_t *)db,
key_part1,
data->key_length,
mtrkrep_key_match_callback,
data);

break;

```

```

    case 'S': /* "STATUS" */
    case 's':

/* : error check status code by translating the string provided into the status code */
        status = MTRK_SC_INVALID;
        mtrkrep_status_string_translate_to_code(key_part1,&status);

/* : make sure we have a valid status code before continuing */
        if (status == MTRK_SC_INVALID) {
            mtrkrep_debug("MTRKREP_KeyList: invalid status code %s\n",key_part1);
            return_code = MTRK_FAIL;
        }
        else {
            mtrkrep_debug("MTRKREP_KeyList: calling KeyFetchByStatus with key %s
(%d)\n",key_part1,status);
            data->original_key = EU_StrDup((void *)&status);
            data->key_length = MTRKDB_STATUS_LEN;
            return_code = MTRKDB_KeyFetchByStatus ((MTRKDB_CONTEXT_t *)db,
                                                    (void *)&status,
                                                    MTRKDB_STATUS_LEN,
                                                    mtrkrep_key_match_callback,
                                                    data);
        }
        break;
    default: /* unknown, cannot continue */
        return_code = MTRK_FAIL;
        break;
}

/* : cleanup and return */
    fs_give((void **)&data->original_key);
    fs_give((void **)&data);
    mtrkrep_debug("MTRKREP_KeyList: function exit\n");
    return(return_code);
}

/* END FUNCTION: MTRKREP_KeyList */

/*

```

```

*/

/* FUNCTION: mtrkrep_rec_match_callback */

/* SYNOPSIS
* Handles a match callback from the MTRKDB_Fetch routines
* END SYNOPSIS */

/* NOTES
* It is the responsibility of the rec_match callback to determine whether it wishes to
* continue to receive records. A non-zero return from this function will cause the fetch
* routine to break out of its loop and return to its caller. IN this case, we have asked
* for a certain number of records to be returned, starting from a specified (full) key.
* This routine checks the number of records returned so far and stops when the number
has
* been reached.
* END NOTES */

static int                                /* R: 0 - return more records, 1 - stop sending
records */
mtrkrep_rec_match_callback (
/* PARAMETERS */
    void * full_key,                      /* I: full key of matching record */
    MTRKDB_DB_TYPE_t db_type,             /* I: what index table the key is from */
    MTRK_STATUS_REC_t * rec,              /* I: record which matched submitted
criteria */
    void * data                           /* I: callback context data */
/* END PARAMETERS */
)
{
/* VARIABLES */
    char **record_string;                  /* record converted to array of strings */
    char *status_string;                   /* status field as string */
    char *timestamp;                       /* timestamp field as string */
    int time_len;                          /* length of time stamp string */
    MTRKREP_REC_MATCH_CB_CONTEXT_t * rec_match_context;
    BOOLEAN match;                         /* whether the full key is a match for the original
partial key */
/* END VARIABLES */

/* : sanity checks */

    mtrkrep_debug("mtrkrep_rec_match_callback: Found record:\n msgid: %s\n status:
%d\n",rec->msgid,rec->status);
    mtrkrep_debug(" host: %s\n to_host: %s\n user: %s\n",rec->host,rec->to_host,rec-
>user);

```

```

    mtrkrep_debug(" description: %s\n",rec->description);

/* : initialization */
    record_string = NULL;
    status_string = NULL;
    timestamp     = NULL;

    rec_match_context = (MTRKREP_REC_MATCH_CB_CONTEXT_t *)data;

/* : debug logging */
    if (db_type == MTRKDB_DB_TYPE_STATUS) {
        mtrkrep_debug("mtrkrep_rec_match_callback: STATUS full_key %d,
original_key %d\n",
                    *(MTRK_STATUS_CODE_t
*)full_key,*(MTRK_STATUS_CODE_t *)rec_match_context->original_key);
    }
    else {
        mtrkrep_debug("mtrkrep_rec_match_callback: MSGID/USER full_key %s,
original_key %s\n",
                    (char *)full_key,rec_match_context->original_key);
    }

/* : first off, make sure the key we received is a match ie: ensure we have not walked past
the key */
    match = mtrkrep_partial_key_match(full_key,rec_match_context->original_key,
                                     rec_match_context->key_length,db_type);

    if (match == FALSE) {
/* : the key does not match, tell callback not to send us any more */
        mtrkrep_debug("mtrkrep_rec_match_callback: keys DO NOT match, exiting
...n");
        return(1);
    }

/* : CLAIM: ok, the key matches */

/* : get storage for return_value */
    record_string = (char **)fs_get(sizeof(char *) * MTRKREP_REC_FIELD_COUNT);
    if (record_string == NULL) {
        return(MTRK_FAIL);
    }

/* : load record into string array */
    if (rec->msgid != NULL) {
        record_string[0] = EU_StrDup (rec->msgid);
    }

```

```

/* : convert status code to a human-readable string */
mtrkrep_status_code_translate_to_string(rec->status,&status_string);
if (status_string != NULL) {
    record_string[1] = status_string;
}
if (rec->host != NULL) {
    record_string[2] = EU_StrDup (rec->host);
}
if (rec->to_host != NULL) {
    record_string[3] = EU_StrDup (rec->to_host);
}
if (rec->user != NULL) {
    record_string[4] = EU_StrDup (rec->user);
}
if (rec->description != NULL) {
    record_string[5] = EU_StrDup (rec->description);
}

/* convert time stamp field to a string */
timestamp = ctime (&(rec->timestamp));

/* convert trailing \n of time stamp string to \0 */
time_len = strlen(timestamp);
timestamp[time_len - 1] = '\0';

if (timestamp != NULL) {
    record_string[6] = EU_StrDup (timestamp);
}
record_string[7] = NULL;

mtrkrep_debug("mtrkrep_rec_match_callback:\n msgid: %s\n status: %s\n host: %s\n",
    record_string[0],record_string[1],record_string[2]);
mtrkrep_debug(" to_host: %s\n user: %s\n description: %s\n timestamp: %s\n",
    record_string[3],record_string[4],record_string[5],record_string[6]);

/* call php callback function */
rec_match_context->php_cb(full_key,record_string,rec_match_context-
>php_cb_data);

/* : cleanup and return */
MTRK_StatusRecDestroy(&rec);

rec_match_context->rec_count++;
if (rec_match_context->rec_count < rec_match_context->pagesize) {
/* : continue to return records */

```

```
        mtrkrep_debug("mtrkrep_rec_match_callback: function exit - continue to return
records\n");
        return (0);
    }
    else {
/* : we've reached the number of records requested, so stop */
        mtrkrep_debug("mtrkrep_rec_match_callback: function exit - no more records
needed\n");
        return (1);
    }
}

/* END FUNCTION: mtrkrep_rec_match_callback */

/*
```



```

*/

/* FUNCTION: mtrkrep_key_match_callback */

/* SYNOPSIS
* Handles a key match from the MTRKDB_KeyFetch routines.
* END SYNOPSIS */

/* NOTES
* It is the responsibility of the key_match callback to determine whether it wishes to
* continue to receive records. A non-zero return from this function will cause the fetch
* routine to break out of its loop and return to its caller. Because the key_match function
* assumes a partial key, it is up to this callback to make sure it is still receiving records
* which satisfy the key and return the correct return code to the caller.
* END NOTES */

static int                                /* R: no return value */
mtrkrep_key_match_callback (
/* PARAMETERS */
    void * full_key,                      /* I: full_key which matched submitted criteria */
    int key_length,                       /* I: length of key returned in bytes */
    MTRKDB_DB_TYPE_t db_type,             /* I: what index table the key is from */
    void * data                           /* I: callback context data */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRKREP_KEY_MATCH_CB_CONTEXT_t *key_match_context; /* the callback
context */
    BOOLEAN match;                                /* whether the full key is a match for the original
partial key */
    char *key_part1;
    char *key_part2;
/* END VARIABLES */

/* : debug */
    mtrkrep_debug("mtrkrep_key_match_callback: function entry\n");

/* : initialization */
    key_match_context = (MTRKREP_KEY_MATCH_CB_CONTEXT_t *)data;
    key_part1 = NULL;
    key_part2 = NULL;

/* : debug logging */
    if (db_type == MTRKDB_DB_TYPE_STATUS) {

```

```

        mtrkrep_debug("mtrkrep_key_match_callback: STATUS full_key %d,
original_key %d\n",
                    *(MTRK_STATUS_CODE_t
*)full_key,*(MTRK_STATUS_CODE_t *)key_match_context->original_key);
    }
    else {
        mtrkrep_debug("mtrkrep_key_match_callback: MSGID/USER full_key %s,
original_key %s\n",
                    full_key,key_match_context->original_key);
    }

/* : first off, make sure the key we received is a match ie: ensure we have not walked past
the key */
    match = mtrkrep_partial_key_match(full_key,key_match_context->original_key,
                                    key_match_context->key_length,db_type);

    if (match == FALSE) {
/* : the key does not match, tell callback not to send us any more */
        mtrkrep_debug("mtrkrep_key_match_callback: keys DO NOT match, exiting
...\n");
        return(1);
    }

/* : CLAIM: ok, the key matches. Now see if it is at the start of a page. */

    mtrkrep_debug("mtrkrep_key_match_callback: keys match, continuing ...\n");
    key_match_context->key_count++;

    mtrkrep_debug("mtrkrep_key_match_callback: key %d, key_count %d, pagesize
%d\n",
                    *(MTRK_STATUS_CODE_t *)full_key,key_match_context-
>key_count,key_match_context->pagesize);

    if ((key_match_context->key_count-1) % key_match_context->pagesize == 0) {
/* : debugging */
        mtrkrep_debug("mtrkrep_key_match_callback: calling php_cb\n",full_key);

/* : split the key into 2 pieces for the return trip */
        MTRKREP_FullKeySplit(db_type,full_key,key_length,&key_part1,&key_part2);

/* : call php callback function */
        key_match_context-
>php_cb(key_part1,key_part2,key_length,key_match_context->php_cb_data);
    }

/* : cleanup and return */

```

```
    mtrkrep_debug("mtrkrep_key_match_callback: function exit, keep sending more  
keys\n");
```

```
/* : keep sending us more keys */
```

```
    return (0);
```

```
}
```

```
/* END FUNCTION: mtrkrep_key_match_callback */
```

```
/*
```

```

*/

/* FUNCTION: MTRKREP_Malloc */

/* SYNOPSIS
* This routine will allocate memory which must be deallocated with MTRKREP_Free
* The memory returned is zeroed.
*
* MTRKREP_Malloc() must not be used until after msadm_init()
*
* The required arguments are:
* size - the size in bytes of the memory to allocate.
*
* END SYNOPSIS */

/* NOTES
* END NOTES */

void *                                /* R: memory block */
MTRKREP_Malloc (                     /* D: allocate memory */
/* PARAMETERS */
    long size                        /* I: bytes to allocate */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
    assert (size > 0);

/* : initialization */

/* : cleanup and return */
    return (fs_get (size));
}

/* END FUNCTION: MTRKREP_Malloc */

/*

```

```

*/

/* FUNCTION: MTRKREP_Free */

/* SYNOPSIS
* This routine will deallocate memory allocated with MTRKREP_Malloc. It does
* not return anything (void), but will set the pointer to NULL.
*
* The required arguments are:
* ptr - the memory being destroyed. This parameter must be the address of
* the pointer to the allocated memory and should be cast to void**
*
* END SYNOPSIS */

/* NOTES
* END NOTES */

void                                /* R: no return value */
MTRKREP_Free (                     /* D: dealloc the specified memory */
/* PARAMETERS */
    void ** ptr                    /* I: memory block to free */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
    assert (ptr != NULL);

/* : initialization */

    fs_give (ptr);

/* : cleanup and return */

}

/* END FUNCTION: MTRKREP_Free */

/*

```

```

*/

/* FUNCTION: mtrkrep_status_string_translate_to_code */

/* SYNOPSIS
* Given a string, translate it to the corresponding status code.
* END SYNOPSIS */

/* NOTES
* END NOTES */

void                                /* R: no return value */
mtrkrep_status_string_translate_to_code (
/* PARAMETERS */
    char *status_string,           /* I: status string to translate */
    MTRK_STATUS_CODE_t *status     /* O: return equivalent status code */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
    if (status == NULL)
        return;

/* : initialization */
    *status = 0;                    /* 0 is not a valid code */

    switch (status_string[0]) {
        case 'S': /* Seen or Submitted */
        case 's':
            if (strcmp(status_string, MTRK_SC_SUBMITTED_STRING) == 0) {
                *status = MTRK_SC_SUBMITTED;
                break;
            }
            if (strcmp(status_string, MTRK_SC_SEEN_STRING) == 0) {
                *status = MTRK_SC_SEEN;
                break;
            }
            break;
        case 'D': /* Delayed or Delivered */
        case 'd':
            if (strcmp(status_string, MTRK_SC_DELIVERED_STRING) == 0) {
                *status = MTRK_SC_DELIVERED;
                break;
            }
    }
}

```

```

    }
    if (strcmp(status_string,MTRK_SC_DELAYED_STRING) == 0) {
        *status = MTRK_SC_DELAYED;
        break;
    }
    break;
case 'N': /* Non DSN Aware */
case 'n':
    if (strcmp(status_string,MTRK_SC_BLACKHOLE_STRING) == 0) {
        *status = MTRK_SC_BLACKHOLE;
    }
    break;
case 'R': /* Rejected or Relayed */
case 'r':
    if (strcmp(status_string,MTRK_SC_REJECTED_STRING) == 0) {
        *status = MTRK_SC_REJECTED;
        break;
    }
    if (strcmp(status_string,MTRK_SC_RELAYED_STRING) == 0) {
        *status = MTRK_SC_RELAYED;
        break;
    }
    break;
case 'F': /* Forwarded */
case 'f':
    if (strcmp(status_string,MTRK_SC_FORWARDED_STRING) == 0) {
        *status = MTRK_SC_DELAYED;
    }
    break;
default: /* unknown */
    break;
}

/* : cleanup and return */
return;
}

/* END FUNCTION: mtrkrep_status_string_translate_to_code */

/*

```

```

*/

/* FUNCTION: mtrkrep_status_code_translate_to_string */

/* SYNOPSIS
* END SYNOPSIS */

/* NOTES
* END NOTES */

void                                /* R: no return value */
mtrkrep_status_code_translate_to_string (
/* PARAMETERS */
    MTRK_STATUS_CODE_t status,      /* I: status code to translate */
    char **status_string            /* OA: return string equivalent of given status code */
/*
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : sanity checks */
    if (status_string == NULL)
        return;

    if (status < MTRK_STATUS_CODE_FIRST || status >
MTRK_STATUS_CODE_LAST) {
        return;
    }

/* : initialization */
    *status_string = NULL;

/* : convert the status code to a string */
    switch (status) {
        case MTRK_SC_SUBMITTED:
            *status_string = EU_StrDup(MTRK_SC_SUBMITTED_STRING);
            break;
        case MTRK_SC_RELAYED:
            *status_string = EU_StrDup(MTRK_SC_RELAYED_STRING);
            break;
        case MTRK_SC_DELIVERED:
            *status_string = EU_StrDup(MTRK_SC_DELIVERED_STRING);
            break;
        case MTRK_SC_FORWARDED:

```



```

        *status_string = EU_StrDup(MTRK_SC_FORWARDED_STRING);
        break;
case MTRK_SC_BLACKHOLE:
    *status_string = EU_StrDup(MTRK_SC_BLACKHOLE_STRING);
    break;
case MTRK_SC_DELAYED:
    *status_string = EU_StrDup(MTRK_SC_DELAYED_STRING);
    break;
case MTRK_SC_REJECTED:
    *status_string = EU_StrDup(MTRK_SC_REJECTED_STRING);
    break;
case MTRK_SC_SEEN:
    *status_string = EU_StrDup(MTRK_SC_SEEN_STRING);
    break;
default:                                /* not one of the recognized cases */
    *status_string = EU_StrDup(MTRK_SC_INVALID_STRING);
    break;
}

/* : cleanup and return */
return;
}

/* END FUNCTION: mtrkrep_status_code_translate_to_string */

/*

```

```

*/

/* FUNCTION: mtrkrep_debug */

/* SYNOPSIS
 * Temporary debug logging system for message tracking API.
 * END SYNOPSIS */

void                                /* R: no return value */
mtrkrep_debug (
/* PARAMETERS */
    char *format,                    /* I: message format */
    ...                               /* I: message arguments */
/* END PARAMETERS */
)
{
/* VARIABLES */
    va_list arg_ptr;                 /* format argument list */
/* END VARIABLES */

/* : don't bother if debug logging not on or file not open */
    if (g_debug_logging == FALSE || g_debug_file == NULL)
        return;

/* : initialize variable argument list handling */
    va_start (arg_ptr, format);

/* : print statement */
    vfprintf(g_debug_file, format, arg_ptr);

/* : finalize variable argument list handling */
    va_end (arg_ptr);

/* : cleanup and return */
    return;
}

/* END FUNCTION: mtrkrep_debug */

/*

```

```

*/

/* FUNCTION: mtrkrep_partial_key_match */

/* SYNOPSIS
* Return TRUE if full_key matches partial key.
* END SYNOPSIS */

/* NOTES
* END NOTES */

static BOOLEAN                                /* R: TRUE if keys match, FALSE
otherwise */
mtrkrep_partial_key_match (
/* PARAMETERS */
    void *full_key,                          /* I: full key to match */
    void *partial_key,                       /* I: partial key to match */
    int partial_key_length,                  /* I: length in bytes of partial key */
    MTRKDB_DB_TYPE_t db_type                /* I: what index table the keys are from */
/* END PARAMETERS */
)
{
/* VARIABLES */
/* END VARIABLES */

/* : first off, make sure the key we received is a match ie: ensure we have not walked past
the key */
    if (db_type == MTRKDB_DB_TYPE_STATUS) {
        if (*(MTRK_STATUS_CODE_t *)full_key != *(MTRK_STATUS_CODE_t
*)partial_key) {
/* : not the same */
            return(FALSE);
        }
        else {
            return TRUE;
        }
    }
    else {
/* : string comparison */
        if (strncmp((char *)full_key,(char *)partial_key,partial_key_length) != 0) {
/* : not the same */
            return(FALSE);
        }
        else {
            return TRUE;
        }
    }
}

```

```
    }  
  
    /* : shouldn't get here */  
    return FALSE;  
}  
  
/* END FUNCTION: mtrkrep_partial_key_match */  
  
/*
```

```

*/

/* FUNCTION: MTRKREP_FullKeyBuild */

/* SYNOPSIS
* Given part one of a key and an alphanumeric record number (part 2), construct a
* a full key value for indexing into the database.
* END SYNOPSIS */

/* NOTES
* The index keys are always comprised of two parts. Part 1 is the data portion of the key
* and is usually a char * but can be binary in the case of the status index key.
* Part 2 is the recno portion of the key and is always binary (an unsigned 64bit integet)
* END NOTES */

void                                /* R: no return value */
MTRKREP_FullKeyBuild (
/* PARAMETERS */
    char * key_type,                /* I: "USER", "MSGID" or "STATUS" */
    char * key_part1,              /* I: the first part of the key, a msgid, user or status
code */
    char * key_part2,              /* I: string containing record number (part 2 of the
key) */
    void **r_full_key              /* OA: part1+part2 concatenated full key to be
returned */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRKDB_RECNO_t recno;          /* to hold record number after conversion
from alphanumeric key_part2 */
    MTRK_STATUS_CODE_t status;     /* status code equivalent of a status string
key value */
    void * full_key_buffer;        /* space to hold key while we are builing it */
    int full_key_length;           /* length of buffer required to hold full key */
    int part1_key_length;          /* length of buffer required to hold part1 of the key
*/
    int recno_offset;              /* offset of recno with the full_key we are building
*/
/* END VARIABLES */

/* : sanity checks */
    if (r_full_key == NULL)
        return;

/* : initialization */

```

```

recno = 0;
status = MTRK_SC_INVALID;
recno_offset = 0;
full_key_buffer = NULL;

switch (key_type[0]) {
    case 'M': /* "MSGID" */
    case 'm':
        mtrkrep_debug("MTRKREP_FullKeyBuild: MSGID %s\n",key_part1);

/* : allocate storage for full key */
        full_key_length = MTRKDB_MSGID_LEN + 1 + MTRKDB_RECNO_LEN;
        full_key_buffer = fs_get(full_key_length);

/* : determine the length of the first part of the key, not allowing it to be largert than the
maximum */
        part1_key_length = (strlen(key_part1) < MTRKDB_MSGID_LEN ?
strlen(key_part1) : MTRKDB_MSGID_LEN);

/* : copy part1 of the key into the first part of the concatenated key */
        memcpy(full_key_buffer,key_part1,part1_key_length);

/* : make sure that part1 of the key is null terminated */
        *((char *)full_key_buffer+part1_key_length) = '\0';

/* : save offset of the recno */
        recno_offset = MTRKDB_MSGID_LEN + 1;
        break;

    case 'U': /* "USER" */
    case 'u':
        mtrkrep_debug("MTRKREP_FullKeyBuild: USER %s\n",key_part1);

/* : allocate storage for full key */
        full_key_length = MTRKDB_USER_LEN + 1 + MTRKDB_RECNO_LEN;
        full_key_buffer = fs_get(full_key_length);

/* : determine the length of the first part of the key, not allowing it to be largert than the
maximum */
        part1_key_length = (strlen(key_part1) < MTRKDB_USER_LEN ?
strlen(key_part1) : MTRKDB_USER_LEN);

/* : copy part1 of the key into the first part of the concatenated key */
        strncpy((char *)full_key_buffer,key_part1,part1_key_length);

/* : make sure that part1 of the key is null terminated */

```

```

        *((char *)full_key_buffer+part1_key_length) = '\0';

/* : save offset of the recno */
    recno_offset = MTRKDB_USER_LEN + 1;
    break;

    case 'S': /* "STATUS" */
    case 's': {
/* : debugging */
        mtrkrep_debug("MTRKREP_FullKeyBuild: STATUS %s\n",key_part1);

/* : translate status string to its numeric equivalent */
        mtrkrep_status_string_translate_to_code(key_part1,&status);

/* if not a valid code, then just exit */
        if (status < MTRK_STATUS_CODE_FIRST || status >
MTRK_STATUS_CODE_LAST)
            return;

/* : allocate storage for full key */
        full_key_buffer = fs_get(MTRKDB_STATUS_LEN+MTRKDB_RECNO_LEN);

/* : put status code into first part of full key */
        memcpy(full_key_buffer,&status,MTRKDB_STATUS_LEN);

/* : save offset of recno */
        recno_offset = MTRKDB_STATUS_LEN;
        break;
    }
    default: /* unknown, cannot continue */
        return;
}

/* : now must translate the recno_string into a MTRKDB_RECNO_t */
if (EU_StrToEC_UINT64 ((char *) key_part2, &recno) == MTRK_FAIL) {
    fs_give((void **)&full_key_buffer);
    return;
}

/* : copy the recno into the part2 of the full_key at the correct offset */
memcpy(full_key_buffer+recno_offset,&recno,MTRKDB_RECNO_LEN);

/* : done, set the full_key output variable and return */
*r_full_key = full_key_buffer;

return;

```

```
}
```

```
/* END FUNCTION: MTRKREP_FullKeyBuild */
```

```
/*
```



```

*/

/* FUNCTION: MTRKREP_FullKeySplit */

/* SYNOPSIS
* Given a full key value retrieved from the database, split the key into two elements:
* a string key part and
* an alphanumeric record number.
* Reverse of FullKeyBuild.
* END SYNOPSIS */

/* NOTES
* END NOTES */

void                                     /* R: no return value */
MTRKREP_FullKeySplit (
/* PARAMETERS */
    MTRKDB_DB_TYPE_t db_type,          /* I: which index table is this key from */
    void * full_key,                   /* I: full key from database */
    int full_key_length,               /* I: length of full_key in bytes */
    char **r_key_part1,                /* OA: returned string containing part1 of the record
key */
    char **r_key_part2                 /* OA: returned string containing part2 of the record
key, an alphanumeric recno */
/* END PARAMETERS */
)
{
/* VARIABLES */
    MTRKDB_RECNO_t recno;               /* to hold record number after conversion
from alphanumeric key_part2 */
    MTRK_STATUS_CODE_t status;          /* status code part of a status full key */
    int recno_offset;                  /* offset of recno within the full key */
    char *status_string;               /* buffer to hold string name of status code */
    char *key_part1;                   /* work buffer for calculating r_key_part1 */
    int part1_key_length;               /* max length in bytes of key_part1 */
    char *alpha_recno;                 /* work buffer to convert recno */
/* END VARIABLES */

/* : sanity checks */
    if (full_key == NULL)
        return;

/* : initialization */
    recno = 0;
    status = MTRK_SC_INVALID;

```

```

    recno_offset = full_key_length - MTRKDB_RECNO_LEN; /* index of start of recno
in full_key */
    part1_key_length = recno_offset - 1; /* subtract 1 more for char key null terminator */
    status_string = NULL;
    key_part1 = NULL;
    alpha_recno = NULL;

/* : get the alpha key part */
    if (db_type == MTRKDB_DB_TYPE_STATUS) {

/* : pull out status code from the beginning part of the full_key */
        memcpy(&status,full_key,MTRKDB_STATUS_LEN);

/* : debugging */
        mtrkrep_debug("MTRKREP_FullKeySplit: STATUS %d\n",status);

/* : if not a valid status code, then just exit */
        if (status < MTRK_STATUS_CODE_FIRST || status >
MTRK_STATUS_CODE_LAST) {
            return;
        }

/* : convert the status code to a string equivalent */
        mtrkrep_status_code_translate_to_string(status,&status_string);

/* : if it's not a valid code, return */
        if (strcmp(status_string,MTRK_SC_INVALID_STRING) == 0) {
            return;
        }

/* : save the status_string as the part1 output var */
        *r_key_part1 = status_string;
    }
    else { /* MSGID or USER */

/* : calculate the maximum length of part1 of the key */
        part1_key_length = (strlen(full_key) < full_key_length ? strlen(full_key) :
full_key_length);

        key_part1 = (char *)fs_get(part1_key_length + 1);
        memset(key_part1,'\0',part1_key_length + 1);

/* : copy part1 of the full_key into its own buffer */
        strncpy(key_part1,full_key,part1_key_length);
        key_part1[part1_key_length] = '\0';

```

```
/* save the string as the part1 output var */
    *r_key_part1 = key_part1;
}

/* : pull out the record number from part2 of the key, using the calculated offset */
    recno = *(EC_UINT64 *)(full_key + recno_offset);

/* : translate the recno into an alphanumeric string */
    EU_StrFromEC_UINT64(recno,&alpha_recno);

/* : save the alphanumeric representation of the recno as the part2 return var */
    *r_key_part2 = alpha_recno;

/* : done, return */
    return;
}

/* END FUNCTION: MTRKREP_FullKeySplit */

/* END MODULE: message_tracking_reporting_api */
```